PATENT ABSTRACTS OF JAPAN

(11)Publication number : 07-287702
(43)Date of publication of application : 31.10.1995

(51)Int.Cl. G06F 17/10
G06F 9/46
G10H 1/00

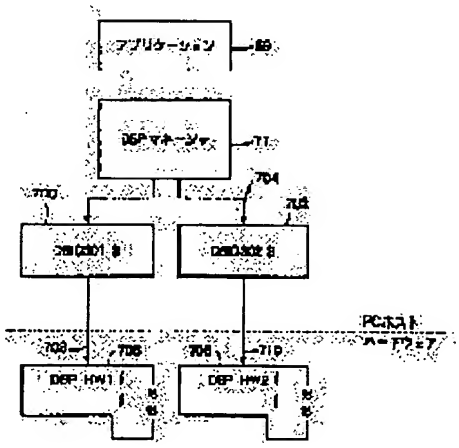
(21)Application number : 07-129869
(22)Date of filing : 31.03.1995

(71)Applicant : INTERNATL BUSINESS MACH CORP <IBM>
(72)Inventor : BAKER ROBERT G
EDUARTEZ JOSE A
HUYNH DUY Q
SWINGLE PAUL R
YONG SUKSOON

(30)Priority
Priority number : 94 220959 Priority date : 31.03.1994 Priority country : US

(54) TASK MANAGEMENT SYSTEM AND METHOD RELATING TO DIGITAL SIGNAL PROCESSOR

(57)Abstract:
PURPOSE: To allow a DSP(digital signal processor) manager to provide a processing to load a task efficiently to a plurality of DSPs by taking various factors such as processing capability, instruction memories in various DSPs located in a data processing system, data and memories into account.
CONSTITUTION: A DSP manager 71 acquires configuration data and relational data relating to a multi-medium hardware device from a BIOS device driver placed inbetween so as to functionally isolate a resource manager from hardware device specific information by means of the software technology. The DSP manager manages a task in execution on the hardware device in the environment of a plurality of DSPs. The DSP manager 71 executes optimizingly assignment of a task of a function and task loading and task elimination without giving interruption to the execution of the task configuring other functions.



(19)日本国特許庁（ＪＰ）　(12)**公開特許公報（A）**　(11)特許出願公開番号

特開平7－287702

(43)公開日　平成７年(1995)10月31日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ	技術表示箇所
Ｇ０６Ｆ　17/10				
9/46	３４０　Ｂ	7737－5B		
Ｇ１０Ｈ　1/00	Ｚ			
			Ｇ０６Ｆ　15/ 31	Ｄ

審査請求　未請求　請求項の数６　ＦＤ　（全 38 頁）

(21)出願番号　特願平7－129869

(22)出願日　平成７年(1995)３月31日

(31)優先権主張番号　２２０９５９

(32)優先日　1994年３月31日

(33)優先権主張国　米国（US）

(71)出願人　390009531
インターナショナル・ビジネス・マシーンズ・コーポレイション
INTERNATIONAL BUSINESS MASCHINES CORPORATION
アメリカ合衆国10504、ニューヨーク州　アーモンク　（番地なし）

(72)発明者　ロバート・ジー・ベイカー
アメリカ合衆国33444－4341　フロリダ州　デルレイ・ビーチ　ノース・ウエスト　ファースト・アベニュー　2112

(74)代理人　弁理士　合田　潔　（外２名）

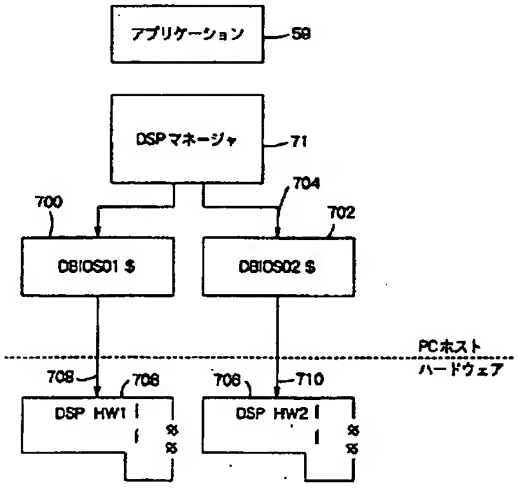
最終頁に続く

(54)【発明の名称】　ディジタル信号プロセッサに関するタスク管理システム及び方法

(57)【要約】

【目的】　DSPマネージャは、処理能力、データ処理システム内に置かれたさまざまなDSP内の命令メモリ、データ・メモリなどの要因を考慮に入れて、複数のDSPへタスクを効率的にロードするための処理を提供する。

【構成】　DSPマネージャが、ソフトウェア技法を介して、ハードウェア装置固有情報から資源マネージャを機能的に隔離するように間に入れられたBIOSデバイス・ドライバから、マルチメディア・ハードウェア装置に関する構成データおよび関連データを取得する。DSPマネージャは、複数DSP環境内のハードウェア装置上で実行中のタスクを管理する。DSPマネージャは、他の機能を構成するタスクの実行に割り込まずに、ある機能のタスクの割当、タスクのロードおよびタスクの除去を最適に実行する。



1

【特許請求の範囲】

【請求項１】各機能が少なくとも１つのタスクから構成される、データ処理システム内の複数のディジタル信号プロセッサによる機能の実行を管理するためのデータ処理システムであって、

データ処理動作用の中央処理装置と、

前記中央処理装置に接続され、通信チャネルによって互いに接続された、複数のディジタル信号プロセッサと、

ディジタル信号プロセッサ・マネージャとを含み、

前記ディジタル信号プロセッサ・マネージャが、

ディジタル信号プロセッサのために存在するプロセッサ資源を識別するための第１識別手段と、

機能がディジタル信号プロセッサにロードされることの表示に応答して、前記機能の一部を構成する、ディジタル信号プロセッサにロードされるタスクを識別するための第２識別手段と、

機能が除去されることの表示に応答して、前記機能の一部を構成する、ディジタル信号プロセッサ上の、ディジタル信号プロセッサから除去されるタスクを識別するための第３識別手段と、

第１識別手段と、追加されるタスクを識別する第２識別手段とに応答して、識別されたタスクの実行をサポートするのに十分な資源を有するディジタル信号プロセッサを選択するための選択手段と、

選択手段によるディジタル信号プロセッサの選択に応答して、ディジタル信号プロセッサ上で実行中の別の機能を構成するタスクのいずれの実行にも割り込まずに、追加すべきタスクを選択されたディジタル信号プロセッサにロードするためのロード手段と、

除去すべきタスクを識別する第３識別手段に応答して、ディジタル信号プロセッサ上で実行中の別の機能を構成するタスクのいずれの実行にも割り込まずに、識別されたタスクをディジタル信号プロセッサから除去するための除去手段とを含む前記ディジタル信号プロセッサ・マネージャとを含むことを特徴とするデータ処理システム。

【請求項２】前記除去手段が、

除去すべき識別されたタスクをフレームから除去する手段と、

除去すべき識別されたタスクの実行を停止させる手段と、

除去すべき識別されたタスクに関連するデータ通信モジュールのそれぞれのメモリ・マークを除去する手段と、

除去すべき識別されたタスクをメモリ・ツリーから除去する手段とを含むことを特徴とする、請求項１のデータ処理システム。

【請求項３】データ処理システム内の複数のディジタル信号プロセッサによるタスクの実行を管理するための前記データ処理システムであって、

データ処理動作用の中央処理装置と、

2

通信チャネルによって互いに接続され、前記中央処理装置に接続された、複数のディジタル信号プロセッサと、

ディジタル信号プロセッサ・マネージャとを含み、

前記ディジタル信号プロセッサ・マネージャは、

ディジタル信号プロセッサによって実行される機能の識別に応答して、機能の一部を構成するタスクをロードのために識別するための第１識別手段と、

機能の一部を構成するロードされたタスクに接続されたデータ通信モジュールが存在するかどうかを識別するための第２識別手段と、

接続されたデータ通信モジュールの存在の識別に応答して、接続されたデータ通信モジュールがリアルタイム・データ通信モジュールであるかどうかを判定する第１判定手段と、

リアルタイム・データ通信モジュールの存在に応答して、識別されたタスクをサポートするのに十分なプロセッサ資源が、データ通信モジュールを含むディジタル信号プロセッサ用に存在するかどうかを判定するための第２判定手段と、

リアルタイム通信モジュールを含む前記ディジタル信号プロセッサ用の十分なプロセッサ資源の存在に応答して、識別されたタスクをディジタル信号プロセッサに追加するための追加手段と、

リアルタイム通信モジュールを含む前記ディジタル信号プロセッサ用の十分なプロセッサ資源の不在またはリアルタイム・データ通信モジュールの不在のいずれかに応答して、通信チャネルが追加のデータ通信モジュールをサポートするのに十分な通信資源を有するかどうかを判定するための第３判定手段と、

追加のデータ通信モジュールをサポートするのに十分な通信資源の存在またはデータ通信モジュールの不在のいずれかに応答して、識別されたタスクをサポートする能力を有する、最大量のプロセッサ資源を有するディジタル信号プロセッサを選択するための手段と、

ディジタル信号プロセッサの選択に応答して、選択されたディジタル信号プロセッサに識別されたタスクをロードするための手段とを含む前記ディジタル信号プロセッサ・マネージャを含み、

１機能を構成するタスクを、ディジタル信号プロセッサ上で実行中の別の機能を構成するタスクの実行に割り込まずに、ディジタル信号プロセッサにロードできることを特徴とするデータ処理システム。

【請求項４】ディジタル信号プロセッサからの除去のため、ディジタル信号プロセッサ上で実行中の機能の一部を構成するタスクを識別するための第４識別手段と、

別の機能を構成するタスクの実行に割り込まずに、ディジタル信号プロセッサから識別されたタスクを除去するための除去手段とをさらに含む、請求項３のデータ処理システム。

【請求項５】前記除去手段が、

識別されたタスクを含むフレームから、識別されたタスクを除去するための手段と、
識別されたタスクの実行を終了させるための終了手段と、
データ通信モジュールへのメモリ・マークを除去するための手段と、
ホストのメモリ・ツリーから識別されたタスクを除去するための手段とを含むことを特徴とする、請求項4のデータ処理システム。
【請求項6】各機能が少なくとも1つのタスクからなり、データ処理システムが、複数のディジタル信号プロセッサに接続された、データ処理動作用の中央処理装置を含み、ディジタル信号プロセッサが、通信チャネルによって互いに接続される、データ処理システム内の複数のディジタル信号プロセッサによる機能の実行を管理するための、データ処理システム内の方法であって、データ処理システムによって実施される、
ディジタル信号プロセッサ用に存在するプロセッサ資源を識別するステップと、
機能がディジタル信号プロセッサにロードされることの表示に応答して、その機能の一部を構成する、ディジタル信号プロセッサにロードされるタスクを識別するステップと、
機能が除去されることの表示に応答して、その機能の一部を構成する、ディジタル信号プロセッサから除去されるディジタル信号プロセッサ上のタスクを識別するステップと、
プロセッサ資源の識別と追加されるタスクの識別とに応答して、識別されたタスクの実行をサポートするのに十分な資源を有するディジタル信号プロセッサを選択するステップと、
選択ステップによるディジタル信号プロセッサの選択に応答して、ディジタル信号プロセッサ上で実行中の別の機能を構成するタスクのいずれの実行にも割り込まずに、選択されたディジタル信号プロセッサに追加すべきタスクをロードするステップと、
除去すべきタスクの識別に応答して、ディジタル信号プロセッサ上で実行中の別の機能を構成するタスクのいずれの実行にも割り込まずに、ディジタル信号プロセッサから識別されたタスクを除去するステップとを含む方法。
【発明の詳細な説明】
【0001】
【産業上の利用分野】本発明は、改良されたデータ処理システムに関し、具体的には、複数のディジタル信号プロセッサ上で実行中のタスクの実行を管理するための方法およびシステムに関する。
【0002】
【従来の技術】ディジタル信号プロセッサ（DSP）は、PC産業で、たとえばデータ圧縮、伸長（解凍（展

開））、電話、音声生成、および、ミキシングや音量制御など他のオーディオ処理などのマルチメディア・リアルタイム要件に合致するための計算力を提供するのに使用されてきた。ハードウェア構成は、通常は、DSP、命令とデータのためのメモリ、および、DAC（ディジタル・アナログ変換器）、ADC（アナログ・ディジタル変換器）、MIDI（Musical Instrument Digital Interface）および電話回線や電話機などの代表的な項目のためのポートなどのハードウェア装置からなる。
【0003】
【発明が解決しようとする課題】DSPタスク・スケジューリングと資源管理を処理するために作成されたプログラムの1つが、米国カリフォルニア州Santa BarbaraのSpectron Microsystems社製のSPOXである。OS/2と同様に、SPOXは、下層のDSPハードウェアに対する高水準ソフトウェア・インターフェースを提供する。しかし、これは、さまざまな単一DSPシステムのための共通インターフェースの作成を目的としている。そのインターフェースは、同種の複数DSPではなく単一DSP（またはマスタ／スレーブ関係のマスタDSP）を目標としているので、負荷平衡化と資源管理への十分な対処が欠けている。
【0004】VCOS（Visible Cache Operating Systemの略）は、AT&T社が自社のDSP 3210ファミリーの部品のために作成したマルチタスク方式の多重処理環境である。VCOSは、ISPOSを用いて最も簡単に識別され、ホスト・オペレーティング・システムのスレーブとなることを目的としている。これは、ホストに緊密に組み込まれるのではないので、IBM社のDSPマネージャが提供するのと同等の範囲の操作性は有しない。これは、IBM社のDSPマネージャの特徴の一部を有するが、具体的には単一DSPサブシステムを目標としている。全体として、Texas Instruments(TI)社、Motorola社およびAT&T社からの既知のDSPサポートは、ハードウェア装置に関する要件の大域管理を提供しない。
【0005】
【課題を解決するための手段】本発明は、マルチメディア・オーディオ・ビデオ・サポートに関するIBM社のディジタル信号プロセッサ・プロジェクトに関する。現在、ディジタル信号プロセッサ（DSP）を管理するソフトウェアは、1つのDSP上のハードウェア・デバイスをサポートする。本発明は、ハードウェア資源をDSPマネージャに集中化することによって、複数のDSP上の複数のハードウェア・デバイスをサポートする。DSP自体を含む各ハードウェア・デバイスは、ハードウェア・デバイス識別子にマッピングされる。各DSP上のすべてのデバイスは、導入可能BIOSデバイス・ドライバによってDSPマネージャに報告される。DSPマネージャは、複数のDSP上のすべてのデバイスを収集し、数を数える。DSPタスクをロードする際に、マネージャは、アプリケーションによって指定されたデバイス識別子によって宛先を突き

止める。プログラム・アプリケーションに影響を及ぼさずに複数のDSPをサポートするために、DSPマネージャを設けて、ハードウェア資源を集中化し、どのDSP上のどの特定のハードウェア・デバイスでも簡単にアクセスできるようにする。

【０００６】具体的に言うと、DSPマネージャは、処理能力、データ処理システム内に置かれたさまざまなDSP内の命令メモリ、データ・メモリなどの要因を考慮に入れて、複数のDSPへタスクを効率的にロードするための処理を提供する。DSPマネージャは、DSP上での他の機能の実行に割り込まずに、選択された機能をロードしたり除去する能力を提供する。

【０００７】

【実施例】ここで図面、具体的には図１を参照すると、コンピュータ１５に電気的に接続された複数のマルチメディア末端装置１３を含むマルチメディア・データ処理システム１１が示されている。当業者は、仕様を参照すれば、コンピュータ１５に、米国ニューヨーク州ArmonkのInternational Business Machines Corporation社製造のPS/2 IBM Computerなど、従来技術で周知のパーソナル・コンピュータ・システムが含まれることを了解するであろう。複数のマルチメディア末端装置１３には、リアルタイム・データまたは非同期ストリーム化データを製作または消費するすべての種類のマルチメディア末端装置が含まれ、CD-ROMプレイヤ１７、マイクロフォン１９、キーボード２１、電話２３、ビデオ・モニタ２５などの末端装置が制限なしに含まれる。マルチメディア末端装置１３のそれぞれは、ストリーム化データを製作または消費するために、マルチメディア・アプリケーション・ソフトウェアによって呼び出される可能性がある。

【０００８】たとえば、CD-ROMプレイヤ１７の動作を、コンピュータ１５に常駐しこれによって制御されるマルチメディア・アプリケーション・ソフトウェアによって制御することができる。CD-ROMプレイヤ１７の出力として生成されたリアルタイム・ディジタル・データ・ストリームを、コンピュータ１５によって、そこに常駐するマルチメディア・アプリケーションの命令に従って受け取り、処理することができる。たとえば、このリアルタイム・ディジタル・データ・ストリームを、従来のコンピュータ・フロッピー・ディスクへ格納するために圧縮したり、通常の電話回線を介してモデム経由で遠隔地にあるコンピュータ・システムに伝送し、このコンピュータ・システムでディジタル・ストリーム化データを解凍（展開）し、アナログ・オーディオ機器で再生することができる。その代わりに、CD-ROMプレイヤ１７からのリアルタイム・データ・ストリーム出力を、コンピュータ１５によって受け取り、ディジタル・フィルタリング、アナログ・フィルタリング、増幅、音声バランスをかけた後に、アナログ信号形式でアナログ・ステレオ・アンプ２９に送り、オーディオ・スピーカ３１および３３に出力することができる。

【０００９】マイクロフォン１９は、環境音に対応するアナログ入力信号を受け取るのに使用できる。このリアルタイム・アナログ・データ・ストリームを、コンピュータ１５に向け、ディジタル形式に変換し、マルチメディア・アプリケーション・ソフトウェアによる操作の対象とすることができる。このディジタル・データは、記憶、圧縮、暗号化、フィルタリング、変換、アナログ・ステレオ・アンプ２９へのアナログ形式での出力、電話２３へのアナログ形式での出力、電話回線への伝送のためのモデムの出力としてディジタル化アナログ形式での提示、ビデオ・モニタ２５での表示用の視覚画像への変換、または、さまざまな他の異なる従来のマルチメディア・ディジタル信号処理動作の対象とすることができる。

【００１０】同様の形で、キーボード２１、電話２３およびビデオ・モニタ２５のアナログまたはディジタルの入出力を、コンピュータ１５内の通常のマルチメディア動作の対象とすることができる。

【００１１】本明細書に記載の発明的貢献から、既存のマルチメディア・データ処理システムに対する多数の大きな長所が明白である。本発明的貢献の１実施例は、（１）無数のユーザ作成マルチメディア・アプリケーションを許容する、オープン・アーキテクチャを提供し、（２）末端装置のリアルタイム動作に干渉せずに、マルチメディア末端装置の調整された動作または同時動作もしくはその両方を可能にし、（３）モジュラー・マルチメディア・ソフトウェア・タスク間でのストリーム化データのリアルタイムまたは非同期の通信を可能にし、（４）コンピュータとマルチメディア末端装置の間でのストリーム化データのリアルタイムまたは非同期の通信を可能にし、（５）マルチメディア・ハードウェア末端装置の通常の機能を仮想化し、ソフトウェアで実行できるようにし、（６）マルチメディア・アプリケーション・ソフトウェアの作成と動作を簡単にする、設計のモジュール性を提供し、（７）他の実行中のマルチメディア・タスクに割り込まずにマルチメディア・タスクの動的再構成を可能にし、したがって、マルチメディア末端装置の一時的なオーバーラップ動作を可能にするマルチメディア・データ処理システムを提供する。

【００１２】図２は、本発明でマルチメディア末端装置１３の動作を制御するマルチメディア・アプリケーションの実行に利用される主なハードウェア構成要素を示すブロック図である。マルチメディア・データ処理動作での通例どおり、中央処理装置（CPU）３３を、コンピュータ１５内に設ける。通常、マルチメディア・アプリケーション・ソフトウェアは、RAMコンピュータ・メモリ３５内に常駐する。CPU３３は、マルチメディア・アプリケーションを構成する命令を実行する。また、マルチ

メディア・データ処理動作での通例どおり、ディジタル信号プロセッサ３７が、補助プロセッサとして設けられ、これは、リアルタイム・データまたは非同期ストリーム化データに対する動作の実行専用である。当業者には周知のとおり、ディジタル信号プロセッサとは、リアルタイム・データに基づく演算またはリアルタイム・データを含む演算の実行専用であり、したがって、マルチメディア末端装置のリアルタイム動作性を許容するように超高速ですばやく応答するように設計されたマイクロプロセッサ・デバイスである。通常、ディジタル信号プロセッサ３７の動作を高速化するために、通常の直接メモリ・アクセス（DMA）３９を設けて、データの高速な取出と記憶を可能にする。本発明では、命令メモリ（IM）４１とデータ・メモリ（DM）４３を別々に設けて、ディジタル信号プロセッサ３７の動作をさらに高速化する。バス４５を設けて、ディジタル信号プロセッサ３７とハードウェア・インターフェース４７の間でデータを通信する。ハードウェア・インターフェース４７には、ディジタル・アナログ変換器とアナログ・ディジタル変換器が含まれる。さまざまなマルチメディア末端装置１３の入出力は、ハードウェア・インターフェース４７のディジタル・アナログ（D/A）変換器とアナログ・ディジタル（A/D）変換器を介して接続される。図２では、例として、電話入出力４９、マイクロフォン入力５３、ステレオ出力の左チャンネル５５および右チャンネル５７が示され、これらは、ハードウェア・インターフェース４７のA/D変換器またはD/A変換器を介して接続される。MIDI入出力もハードウェア・インターフェース４７からディジタル信号プロセッサ３７に接続されるが、これはA/D変換器やD/A変換器には接続されない。

【００１３】図３は、マルチメディア・ディジタル信号処理動作の主なソフトウェア構成要素を示すブロック図である。この図の上部は、CPU３３（図２）内で処理されるソフトウェアを表し、この図の下部は、ディジタル信号プロセッサ３７（図２）内で処理されるソフトウェア７３を表す。図からわかるように、マルチメディア・アプリケーション５９は、マルチメディア・アプリケーション・プログラミング・インターフェース（API）６１を介して、電話応答機（TAM）ドライバ６３、サウンド・ドライバ６５、モデム・ドライバ６７および他のディジタル信号処理ドライバ６９などのデバイス・ドライバを経てDSPマネージャ７１と通信する。DSPマネージャ７１は、複数のモジュラー・マルチメディア・ソフトウェア・タスクのうちのどれでも要求でき、実行のためディジタル信号プロセッサ３７（図２）にロードすることができる。TAMタスク７５、サウンド・タスク７７およびモデム・タスク７９を含む、複数の代表的なモジュラー・タスクを図３に示す。他のさまざまなディジタル信号処理タスクは、「他のDSPタスク」と記されたブロック８１によって表される。これらのマルチメディア・ソフトウェア・タスクは、マルチメディア末端装置または他のモジュラー・マルチメディア・ソフトウェア・タスクによって消費または製作されるリアルタイム・データまたは非同期ストリーム化データに対して実行される、実質的に削減できない動作を表す。マルチメディア・ソフトウェア・タスクは、ディジタル信号プロセッサによって実行可能な命令の組と、他のモジュラー・マルチメディア・ソフトウェア・タスクまたはマルチメディア末端装置によって作成されたか他のモジュラー・マルチメディア・ソフトウェア・タスクまたはマルチメディア末端装置によって消費されるリアルタイム・データ点または非同期データ点を表すデータ点の組とを表す。マルチメディア・アプリケーション５９は、複数のマルチメディア・モジュラー・ソフトウェア・タスクによって表される動作の実行を頻繁に要求する。この場合、タスクは、１モジュール内に配置されるが、そのさまを図１０に示す。

【００１４】複数のDSPを使用するデータ処理システムのブロック図である図４を参照すると、複数の異なる通信チャネルを使用して、DSPを相互接続し、DSP間の通信を実現することができる。本発明の好ましい実施例によれば、DSP1、DSP2およびDSP3は、バス６８に接続される。バス６８は、バス・インターフェース・チップ（BIC）７０に接続され、BIC７０は、PCホストの主バス（すなわちマイクロチャネル）に接続される。「マイクロチャネル」は、International Business Machines Corporation社の登録商標である。BIC７０は、PCホストとDSPハードウェアの間の通信を制御する。BIC７０とDSPを接続するバス構成とに関する情報は、米国特許出願通し番号第０８／１５５３１１号明細書等にある。

【００１５】図示の実施例には、DSP間の接続と通信をもたらす「ローカル・バス」が図示されているが、「ローカル・バス」以外の多数の異なる種類の通信チャネルを使用して、DSP間の通信を実現するためにDSPを相互接続することができる。たとえば、共用メモリ・システムを、大域共用メモリ構成または二重ポート式共用メモリ構成で実施することができる。さらに、DSP間の通信は、さまざまなDSP内の直列ポートを相互接続する直列ポート配置を介して実現できる。たとえば、４個のDSPを使用し、各DSPが少なくとも３つの直列ポートを含む場合、４個のDSPのすべてを、直列ポートを介して互いに相互接続することができる。

【００１６】DSPマネージャの大きな特徴が、装置独立な形で複数DSPサポートを提供する能力である。この目標は、DSP BIOSインターフェースの使用を介して達成される。図５のDSP BIOSインターフェース７０４を参照されたい。このインターフェースは、ハードウェア固有の情報をDSPマネージャ７１から隔離する。これによって、他のアプリケーションに影響を与えずに、カードを挿入するという簡単な処理によってハードウェアを含め

ることができるようになる。

【００１７】下記は、それぞれのDSP BIOSxドライバを介してDSPマネージャ７１によって管理される機能のリストである。

・BIOS_Read

DSPのデータ・メモリまたは命令メモリから指定されたバッファにnワードを読み取る。

・BIOS_Write

DSPのデータ・メモリまたは命令メモリに指定されたバッファからnワードを書き込む

・BIOS_Query_Info

MIPs、メモリ・サイズ、ハードウェア・デバイスなどのDSP情報を問い合わせる。

・BIOS_Reset

指定されたDSPをリセットする。

・BIOS_Halt

指定されたDSPをホールト（halt）する。

・BIOS_SetIPC

IPCルーチン・アドレスをセットする。

DSP BIOSデバイス・ドライバを導入するステップは、次のとおりである。

１．デバイス・ドライバは、当初は名前"DBIOS00$"を含む。

２．初期設定中に、デバイス・ドライバは、デバイス・ドライバ"DBIOS01$"にATTACHDDを発行する（図５のDBIOSxドライバ７００を参照されたい）。これが成功した場合、そのデバイス・ドライバは、"DBIOS01$"を"DBIOS02$"に変更し（図５のDBIOSxドライバ７０２を参照されたい）、同一の動作を実行する。このステップを、エラーが発生するまで繰り返す。

３．デバイス・ドライバは、"DBIOS00$"をエラーになった番号、たとえば"DBIOS10$"に変更する。

４．デバイス・ドライバは、直前のデバイス・ドライバのIDC入口点をセーブし、終了する。

【００１８】DSPマネージャが走行する時、その初期設定コードは、エラーになるまで複数のDOSOPENを呼び出すことによって、最後のDSP BIOSドライバを突き止める。その後、DSPマネージャは、最後のBIOSドライバにIOCTLコマンドを発行し、最後のBIOSドライバは、同じコマンドを直前のデバイス・ドライバに発行し、このデバイス・ドライバが、同じ動作を行う。この処理の結果、DSPマネージャは、導入されたすべてのBIOSドライバの入口点のすべてを得る。

【００１９】DSPハードウェア表現

この実施例のために選択された抽象の水準は、デバイスIDと称する数値を使用して、各ハードウェア・デバイスを識別することである。デバイスIDは、下記の３２ビット・ワードである。

ビット（上位ﾜｰﾄﾞ） ビット

3 1

1 6

TTDD DDDD DDDD DDDD

T=００、 入力デバイス

０１、 出力デバイス

１０、 入出力デバイス

D=デバイス・タイプ

ビット （下位ワード） ビット

1 0

5 0

NNNN NNNN NNNN NNNN

N=デバイス順序数

０ － 省略時

１ － デバイス1

２ － デバイス2

…

など

現在定義されているハードウェア・デバイスIDは、下記の通りである。

上位ワード（１６進）	デバイス名
0001	ステレオ・ライン入力
8001	ステレオ・ライン出力
E006	電話回線
0006	電話機
0008	MIDI入力
8008	MIDI出力
E009	DSP
E00A	UART

【００２０】DSP BIOSドライバ導入の最後に、DSPマネージャは、BIOS_Query_Infoコマンドを発行して、各DSPの下記の資源情報を得る。

```
STRUCTURE /* DSP情報 */
USHORT DSP_MIPSs; /* DSP MIPs数 */
USHORT DSP_DataStore; /* KW単位のデータ記憶サイズ */
USHORT DSP_InstStore; /* KW単位の命令記憶サイズ */
USHORT DSP_Smart_Cable_ID; /* 0FFH:使用不能 */
USHORT DSP_Slot_Number; /* アダプタのスロット番号 */
USHORT DSP_Adapter_ID; /* アダプタID */
USHORT DSP_COM; /* 使用するCOMポート、ビット0 = 1 = COM1 */
    /*ビット1 = 1 = COM2、など */
USHORT DSP_NumHWs /* ユニークなハードウェアの数 */
```

```
ULONG DSP_HWID[SDP_NumHWs];
/* カウントの代わりに使用される順序数 */
/* たとえば、0002 8001 = 2ステレオ出力ライン */
.
.
END
```

USHORTは、符号なし短整数（１６ビット）を意味し、ULONGは、符号なし長整数（３２ビット）を意味する。

【００２１】その後、DSPマネージャは、スロット番号に基づいてハードウェア・デバイスに番号をつける。たとえば、スロット１のステレオ出力ラインには、STEREO-OUT-1という番号をつけ、スロット２のステレオ出力ラインには、STEREO-OUT-2という番号を付け、この方式を繰り返す。

【００２２】アプリケーションがDSPタスクをロードするためにDSPマネージャAPIを発行する時には、宛先を示すためにハードウェア・デバイスIDも指定する。

【００２３】図５は、DSPマネージャ７１が、デバイス識別子を使用してハードウェアから独立に複数のDSPデバイスを管理する方法を示す図である。

【００２４】DSPマネージャ７１は、DSP BIOSインターフェース７０４を使用して、例の複数のDSPのDBIOSxドライバ７００および７０２を接続し、維持する能力を有する。DSPマネージャ７１は、図５のDSP HW1ユニットおよびDSP HW2ユニット（ハードウェア・ユニット７０６）からなるハードウェア資源とソフトウェア資源を、この実施例のデバイス識別子とDSP BIOSインターフェース７０４を介して維持し、割り振り、割振り解除する。

【００２５】個々のDBIOSxドライバ７００および７０２のそれぞれは、ハードウェア・ユニット７０６へのそれ自体のインターフェース７０８および７１０を有する。DSPBIOSインターフェース７０４は、DSPマネージャ７１およびマルチメディア・アプリケーション５９を、Mwaveハードウェアなどの異なるタイプのハードウェア・ユニット７０６から「隔離」する共通インターフェースを提供する。

【００２６】図６は、DSPマネージャ７１とDBIOSxドライバ７００および７０２の間のDSP BIOSインターフェース７０４の使用と、ハードウェア・ユニット７０６へのDSPタスクのロードを説明する論理流れ図である。

【００２７】BIOS_QUERY_INFO７１２コマンドを発行して、DSPに関する情報を見つける。DSP_INFO構造体に記述されたこの情報は、下記の決定に使用される。

- DSP_MIPS　DSPの処理能力
- DSP_DataStore　使用可能なデータ記憶容量
- DSP_InstStore　使用可能な命令記憶容量
- DSP_Slot_Number　アダプタのスロット番号
- DSP_Adapter_ID　アダプタのタイプ
- DSP_COM　使用するCOMポート
- DSP_NumHWs　ユニークなハードウェアの数
- DSP_HWID[DSP_NumHWs]　本発明で指定されるハードウェア・デバイスID

【００２８】BIOS_QUERY_INFO７１２から返された情報を使用して、ブロック７１４で、DSPタスクをロードするのに十分な資源と適切なタイプの資源があるかどうかの判定を行うことができる。ここで説明する処理は、DSPマネージャ７１とDBIOSxドライバ７００および７０２の間の相互作用であることに留意されたい。また、DSPマネージャ７１は、マルチメディア・アプリケーション５９を、そのシステムに存在するハードウェア・ユニット７０６が何であり、DSPカードの枚数がいくつであるかという実際の低水準の選択から「隔離」することに留意されたい。

【００２９】次のステップは、DSPタスクをロードする位置に関して、次に使用可能な命令記憶アドレスとデータ記憶アドレスを見つけるためのBIOS_READコマンド７１６の発行である。ステップ７１８で検査を行って、DSPコードをロードする位置が存在することを検証する。

【００３０】BIOS_WRITEコマンド７２０を複数回発行して、DSP用のデータ・メモリ４３および命令メモリ４１にDSPタスクをロードすることができる。成功をステップ７２２で検証する。これらの成功およびエラーのすべてが、ハードウェアおよびソフトウェアの資源能力の現在の状態を維持するという本発明の技法のDSPマネージャ７１の実施態様により、最終的にマルチメディア・アプリケーション５９に返されることに留意されたい。

【００３１】BIOS_SETIPC７２４を発行して、DBIOSxドライバ７００または７０２とDSPマネージャ７１の間の割込みコールバック（IPC）アドレスをセットアップする。図５のDSP HW1およびDSP HW2のために使用できるDSPハードウェア・カードあたりの割込み「チャネル」の数は有限なので、ステップ７２６でチャネルが使用可能であるかどうかを検証する。

【００３２】最後に（任意指定）BIOS_READ７２８を行って、ステップ７３０で、DSPマイクロコードの正しさを確認する。

【００３３】ステップ７３２とステップ７３４でのBIOS_WRITEは、DSPタスクの走行を開始するために発行される。この時点で、タスクとマルチメディア・アプリケーション５９の間のすべての通信は、DBIOSxドライバ７００または７０２によって処理される割込みを介し、DSPマネージャ７１に渡されるデバイス識別を介して行われる。

【００３４】図７は、DBIOSxドライバ７００または７０２とDSPマネージャ７１の間の割込み処理の論理の流れを示す図である。

【００３５】この状況では、タスクが割り込まれており、DBIOSxドライバが、BIOS_SETUPコマンドによって確立された割込みハンドラにその割込み呼出しを渡している。図７の流れの処理は、DSPマネージャ７１による割込みの維持を示すものである。

【００３６】BIOS_READコマンド７３６をハードウェア・ユニット７０６に発行して、ハードウェア割込みベクトルを読み取る。ステップ７３８で検証を行って、割込みをかけたタスクとハードウェアの組合せの所有権を判定する。ステップ７３８での検証が確立されたならば、BIOS_WRITEステップ７４０が発生し、コマンドを発行して、この特定のタスクとハードウェアの組合せに関する割込みベクトルをクリアする（ステップ７４２）。

【００３７】ステップ７４４でタスクからBIOS_READを発行して、割込みが発行された理由を判定する。ステップ７４６で、この割込みがハードウェア障害のために発行されたと示される場合、ステップ７４８のBIOS_HALTとステップ７５０のBIOS_RESETを発行して、ディジタル信号プロセッサ上のハードウェアを停止させ、リセットしなければならない。

【００３８】ステップ７５２で、この割込みが正常なタスク維持のために発行された（すなわち、次のデータや正常完了など）場合、ステップ７５４でBIOS_READを発行して、タスクと資源の状態に関する情報を得る。ステップ７５６で、１回以上のBIOS_Writeを発行して、正しい動作のためにこのタスクにサービスする。

【００３９】図８は、本明細書に記載の、DSP BIOSドライバ初期設定の流れ図と、デバイス識別を使用して複数のDBIOSxドライバをシステムにロードする方法を示す図である。

【００４０】DSPマネージャ７１は、初期設定の際にDBIOS00$識別子から開始する。ドライバをロードしようとする時に、DSPマネージャ７１は、ステップ７５８で識別子をDBIOS01$（DBIOSxドライバ７００）にセットし、ステップ７６０でこのドライバにATTACHDDを発行する。このATTACHDDコマンドによって、ドライバの入口点がセットされ、ドライバへの通信「経路」が確立される。ATTACHに成功した場合、DBIOSxドライバが存在し、ステップ７６２でXXを増分して、次のドライバに関してステップ７６０のATTACHDDを再試行する。

【００４１】このループは、ステップ７６４でATTACHDDが失敗するまで繰り返される。失敗の場合、ステップ７６６で、前のXXをそのシステム内の新しいドライバのためにセーブする。この時点で、DSPマネージャ７１は、DBIOSxドライバのそれぞれとの初期通信を有することに留意されたい。

【００４２】図９を参照すると、たとえばDBIOSxドライバ７００および７０２などのドライバのすべてがステップ７６０によって互いに成功裡に接続(ATTACH)されたならば、DSPマネージャ７１は、これらのドライバのアドレスを登録する。これは、DSPマネージャ７１が、ブロック７６８で最初のドライバから開始し、ステップ７７０でドライバに対する連続的なオープンを発行することによって行われる。

【００４３】ステップ７７０でのオープンのそれぞれによって、たとえばDBIOSxドライバ７０２などのDBIOSxxドライバが、それ自体を初期設定する。ステップ７７０でのドライバの連続的なオープンは、ステップ７７２でドライバの存在を確立する方法でもあることに留意されたい。ステップ７７４で、xxレベルを増分する。

【００４４】すべてのドライバが初期設定されたならば、ステップ７７６で最後にオープンされたドライバにQuery_IDCコマンドを発行して、DBIOSxドライバ７００および７０２の間の照会のカスケード効果を引き起こし、ステップ７６０からのIDC入口点のすべての登録という最終結果をもたらす。

【００４５】本明細書で述べるように、DSPマネージャ７１とDBIOSxドライバ７００および７０２の間で発行される、ステップ７７０からのドライバに対するオープンとステップ７６０でのATTACHDDのすべてが、構成要素間でのBIOS通信リンクまたはDSP BIOSインターフェース７０４を形成する。さらに、デバイス識別技法とハードウェア資源識別技法を使用することによって、通信のすべてがマルチメディア・アプリケーション５９から透過的になる。

【００４６】図１０は、N個のモジュラー・マルチメディア・ソフトウェア・タスク８５および１０３のグループ化を表すモジュール８３からなるマルチメディア動作を示すブロック図である。図からわかるように、タスク１は、複数のコード・セグメント８７、８９、９１および９３と、複数のデータ・セグメント９５、９７、９９および１０１からなる。タスク１０３は、同様に、複数のコード・セグメント１０５、１０７、１０９および１１１と、複数のデータ・セグメント１１３、１１５、１１７および１１９からなる。図２をもう一度参照すると、タスク８５および１０３のコード・セグメントは、命令メモリ４１に常駐し、タスク８５および１０３のデータ・セグメントは、データ・メモリ４３に常駐する。図２に視覚的に示されているように、マルチメディア末端装置１３によって作成または消費されるリアルタイム・データまたは非同期データは、バス１２１を介してデータ・メモリ４３に直接読み書きすることができる。

【００４７】物理的な接続を、図１１に詳細に示す。図１１は、図１に示されたマルチメディア末端装置１３と図２に示された専用のデータ・メモリ４３の間でのリアルタイム・データまたは非同期ストリーム化データの直接読み書きを示すブロック図である。図からわかるよう

に、電話入出力４９、マイクロフォン入力５３、MIDI端子入出力５１およびステレオ出力の左チャンネル５５および右チャンネル５７は、直列にまたは図２のハードウェア・インターフェース４７内のさまざまなアナログ・ディジタル（A/D）変換器およびディジタル・アナログ（D/A）変換器とバス１２１とを介してデータ・メモリ４３へ通信する。複数のデータ通信モジュール１３７をデータ・メモリ４３内に設けて、ディジタル信号プロセッサ３７とマルチメディア末端装置１３の間でリアルタイム・データまたは非同期ストリーム化データを受け渡しする。データ通信モジュールのそれぞれに、下で詳細に説明する環状メモリ・バッファが含まれ、この環状メモリ・バッファに、電話入力環状バッファ１２３、電話出力環状バッファ１２５、マイクロフォン入力環状バッファ１２７、左チャンネル・ステレオ出力環状バッファ１２９、右チャンネル・ステレオ出力環状バッファ１３１、MIDI入力環状バッファ１３３およびMIDI出力環状バッファ１３５が含まれることが好ましい。図１１からわかるように、電話入力４９、マイクロフォン入力５３およびMIDI端子入力５１からのストリーム化データは、通常の直接メモリ・アクセス（DMA）動作の使用を介してデータ・メモリ４３に向けられる。

【００４８】図３をもう一度参照すると、複数のモジュラー・マルチメディア・ソフトウェア・タスクが、ディジタル信号処理BIOSに常駐するが、これには、直接メモリ・アクセス・インターフェース・タスク１３９、汎用非同期送受信器（UART）インターフェース・タスク１４１、アナログ変換インターフェース・タスク１４３およびCDディジタル・アナログ変換（CD-D/A）インターフェース・タスク１４５が含まれる。TAMタスク７５、サウンド・タスク７７、モデム・タスク７９および他のDSPタスク８１などのマルチメディア・モジュラー・ソフトウェア・タスクは、データ・メモリ４３と複数のマルチメディア末端装置１３の間でのリアルタイム・データまたは非同期ストリーム・データの読み書きのために提供されるものなどのデータ通信モジュールの使用を介して、お互いの間およびディジタル信号処理BIOSと、リアルタイム・データまたは非同期ストリーム化データを通信する。

【００４９】マルチメディア動作用のモジュール化されたオープン・アーキテクチャは、図１２ないし図２５を参照して下で説明する、新規のデータ通信モジュールをタスク間制御ブロックと組み合わせて使用することを介して得られる。図１２、図１３および図１４は、モジュール化されたオープン・アーキテクチャを得るためのデータ通信モジュールとタスク間制御ブロックの使用を表すブロック図である。図１２は、どちらも単一のディジタル信号プロセッサに常駐するタスク１５１からタスク１５３への連続的なリアルタイム単一方向データ転送の実現に使用することができるデータ通信モジュール１５５を示す図である。タスク間制御ブロック（ＩＴＣＢ）１５７は、タスク１５１内で定義され、タスク間制御ブロック（ＩＴＣＢ）１５９は、タスク１５３内で定義される。タスク間制御ブロック１５７および１５９は、タスク１５１および１５３の間で状況と制御情報を渡すために設けられる。

【００５０】図１３は、タスク１５１とタスク１５３の間への暗号化タスク１６１の挿入を示す図である。図からわかるように、データ通信モジュール１５５は、タスク１５１から暗号化タスク１６１にデータを渡す。データ通信モジュール１６３を暗号化タスク１６１とタスク１５３の間に設けて、暗号化タスク１６１からタスク１５３への連続的なリアルタイム単一方向データ伝送を可能にする。図からわかるように、タスク間制御ブロック１５７および１５９は、やはりタスク１５１とタスク１５３の間での状況と制御情報の受け渡しに使用される。タスク１５１、タスク１５３および暗号化タスク１６１は、すべてが単一のディジタル信号プロセッサに常駐する。

【００５１】図１４は、第１のディジタル信号プロセッサに常駐するタスク１５１およびタスク１６１と、第２のディジタル信号プロセッサに常駐するタスク１５３を有し、これらのタスクがマルチメディア・タスクを実行するために協力している構成を示す図である。図からわかるように、データ通信モジュール１５５は、タスク１５１とタスク１６１の間のデータの受け渡しをもたらす。データ通信モジュール１６３は、タスク１６１からタスク１５３へのデータ渡しをもたらす。追加のタスク間制御ブロック１６５が、タスク１５１とタスク１６１の間での状況と制御情報の受け渡しをもたらし、タスク間制御ブロック１５７および１５９が、タスク１５１とタスク１５３の間の状況と制御変数の受け渡しを行う。

【００５２】図１２、図１３および図１４に示された形で、モジュラー・マルチメディア・ソフトウェア・タスクとマルチメディア末端装置を、無数のユーザ選択の組合せで互いに接続することができる。ユーザは、個々の必要に応じて独自のタスクをコード化した後に、データ通信モジュールとタスク間通信ブロックを使用することによって、これらのタスクを先在するモジュラー・マルチメディア・ソフトウェア・タスクに接続することができる。ユーザは、モジュラー・マルチメディア・ソフトウェア・タスクの特定のコーディングに配慮する必要はなく、データ通信モジュールとタスク間制御ブロックを介するデータ受け渡しのデータ通信プロトコルだけを考慮すればよい。

【００５３】したがって、本発明は、ユーザが、事前にパッケージ化されたモジュラー・マルチメディア・ソフトウェア・タスクの動作に対する過度の危険なしに、ユーザ自身のマルチメディア・アプリケーション・ソフトウェアをコーディングできるようにする、比較的危険性

の低いオープン・アーキテクチャを提供する。データ通信モジュールとタスク間通信ブロックを使用することによって、非常に強化された接続性がもたらされ、実質的にオープンなアーキテクチャが提示され、したがって、非常に多数のソフトウェア・ベンダが、他のソフトウェア・ベンダならびにエンド・ユーザ自身によって生成された他の事前にパッケージ化されたマルチメディア・アプリケーション・ソフトウェアに干渉せずにディジタル信号プロセッサによって実行される広範囲のモジュラー・マルチメディア・ソフトウェア・タスクを含む事前にパッケージ化されたマルチメディア・アプリケーション・ソフトウェアを作成し、販売することが可能になる。この実質的にオープンなアーキテクチャを支持するためには、データ通信モジュールとタスク間通信ブロックに関して複数の厳密な規則が必要である。

【００５４】図１５、図１６および図１７は、モジュラー・マルチメディア・ソフトウェア・タスクとマルチメディア末端装置の間でのデータ転送のためにデータ通信モジュールを秩序だった形で使用できるようにする好ましい動作規則を示す図である。この好ましい実施例では、データ通信モジュールが、２つ以上のモジュラー・マルチメディア・ソフトウェア・タスクの間または１つのモジュラー・マルチメディア・ソフトウェア・タスクと１つまたは複数のマルチメディア末端装置の間でデータ・ストリームを受け渡しするのに使用される環状バッファである。図１５からわかるように、環状メモリ・アレイ１６７は、複数のバイトまたはワードからなり、これらは、メモリ・セグメント１６９などの環状メモリ・アレイ１６７内のセグメントによって図示されている。図１５、図１６および図１７では、斜線のはいったメモリ・セグメントに新しいデータが含まれ、斜線のないメモリ・セグメントには新しいデータが含まれない。

【００５５】リアルタイム・データまたは非同期データの秩序だった通信を確保するために、あるタスクを、特定のデータ通信モジュールの「オーナー」として指定する。あるデータ通信モジュールは、１つのオーナーだけを有することができる。オーナーは、そのデータ通信モジュールの環状メモリ・アレイ１６７への書込を許可される唯一のタスクである。オーナー・タスクは、データが最後に書き込まれたメモリ・セグメントを識別する「オーナー・ポインタ」または書込ポインタ１７１を制御する。１つまたは複数のマルチメディア・ソフトウェア・タスクは、データ通信モジュールの「ユーザ」として識別される。「ユーザ」タスクまたは末端装置は、データが最後に読み取られた環状メモリ・アレイ１６７内のメモリ・セグメントを識別するユーザ・ポインタまたは「読取ポインタ」１７３を制御する。オーナー・タスクが環状メモリ・アレイ１６７にデータを書き込む前に、書込ポインタ１７１を増分して、環状メモリ・アレイ１６７内の次の連続したメモリ・セグメントを識別する。ユーザ・タスクが環状メモリ・アレイ１６７からデータを読み取る前に、読取ポインタ１７３を同様に増分して、次に読み取る環状メモリ・アレイ１６７内のメモリ・セグメントを識別する。データは、環状メモリ・アレイ１６７に対して１方向にのみ読み書きされる。これを、図１５の矢印１７５によって示す。

【００５６】図１６は、「空」のデータ通信モジュールを示す図である。オーナー・タスクによって環状メモリ・アレイ１６７に書き込まれたデータのすべてが、ユーザ・タスクによって読み取られたか「消費」されている。この状況では、書込ポインタ１７１と読取ポインタ１７３が、同一のメモリ・セグメントを識別している。図１７は、「満杯」のデータ通信モジュールを示す図である。この状況では、オーナー・タスクが、環状メモリ・アレイ１６７内の使用可能なメモリ・セグメントのすべてにデータを書き込んでいる。この状況は、書込ポインタ１７１が読取ポインタ１７３より１メモリ・セグメント後ろにある時に識別できる。読取ポインタ１７３または書込ポインタ１７１を増分すると、そのポインタは、矢印１７５の方向にバッファ内を前進する。

【００５７】リアルタイム・データまたは非同期ストリーム化データの通信を簡単にする上で、４つの標準化通信プロトコルが、本発明の好ましい実施例によるデータ通信モジュールと共に使用される。図１８、図１９、図２０および図２１は、本発明の好ましい実施例でデータ通信モジュールと共に使用される４つの標準化通信プロトコルを表す図である。図１８は、同期式プロトコルを表す。図１９は、オーナー・データ駆動プロトコルを表す。図２０は、ユーザ・データ駆動プロトコルを表す。図２１は、安全データ駆動プロトコルを表す

【００５８】図１８に示された同期式プロトコルを参照すると、オーナー・タスク１７７は、一定の速度で環状メモリ・アレイ１６７にデータを書き込み、ユーザ・タスク１７９は、同じ一定速度でデータを読み取る。したがって、オーナー・タスク１７７とユーザ・タスク１７９の両方が、ユーザ・タスクに提供される書込ポインタ１７１の位置（アドレス"PUTP'）に関するフィードバックもオーナー・タスクに提供される読取ポインタ１７３の位置（アドレス"GETP'）に関するフィードバックもない、「開ループ」で走行する。オーナー・タスク１７７は、ユーザ・タスク１７９が正しい速度でデータを消費していると仮定し、ユーザ・タスク１７９は、オーナー・タスク１７７が正しい速度でデータを作成していると仮定する。オーナー・タスク１７７とユーザ・タスク１７９のどちらもが、バッファの「空」状態や「満杯」状態を検査せず、オーナー・タスク１７７とユーザ・タスク１７９のどちらもが、もう一方のタスクのポインタの位置またはアドレスを知らない。したがって、オーナー・タスク１７７とユーザ・タスク１７９は、書込ポインタ１７１と読取ポインタ１７３の均一な間隔（時間上

の）によって明示される正確な同期の維持を保証するために、同一のハードウェア割込み供給源から同一のサンプル速度で走行することが重要である。同期式プロトコル・モードの動作で動作しているデータ通信モジュールにデータを書き込めるオーナー・タスクは1つだけであるが、任意の個数のユーザ・タスクをデータ通信モジュールに接続することができる。

【0059】図19は、オーナー・タスク177が、それ自体の速度で環状メモリ・アレイ167にデータを書き込むオーナー・データ駆動プロトコルを示す図である。ユーザ・タスク179は、オーナー・タスク177に遅れずについてゆき、オーナー・タスク177が生成し環状メモリ・アレイ167に書き込むデータのすべてを消費することが期待される。オーナー・タスク177は、「開ループ」で走行し、読取ポインタ173の位置（すなわちアドレス）に関するフィードバックを全く有さず、環状メモリ・アレイ167が「満杯」や「空」であるかどうかの判定を全く行わない。オーナー・タスク177は、一定速度または、所定の最大速度までの可変速で、環状メモリ・アレイ167にデータを書き込むことができる。この最大速度は、下で詳細に説明するように、最大ワード毎フレーム（MWPF）単位で、データ通信モジュールを定義する「マクロ」で指定される。環状メモリ・アレイ167のサイズも、そのマクロによって"SIZE"として指定される。サイズ（SIZE）、最大ワード毎フレーム（MWPF）ならびに書込ポインタ171のアドレス(APUT)は、すべてがユーザ・タスク179によって維持され、ユーザ・タスク179がオーナー・タスク177によって製作されたデータのすべてを消費することを保証するのに使用される。ユーザ・タスク179は、環状メモリ・アレイ167が絶対に満杯にならない、すなわち、オーバーフローしないことを保証する責任を負う。ユーザ・タスク179が遅れをとると、オーナー・タスク177が新データで旧データを上書きすることになる。通常、ユーザ・タスク179は、使用可能なデータのすべてを読み取るか、1ブロック分のデータが使用可能になると同時にそのブロックのデータを読み取ることによって、環状メモリ・アレイ167を空にする。上で述べたように、ユーザ・タスク179は、オーナー・タスク177の書込ポインタ171のアドレスを知っており、したがって、ポインタの位置を比較することができる。書込ポインタ171のアドレスPUTPが読取ポインタ173のアドレスGETPと等しい場合、環状メモリ・アレイ167は「空」である。そうでない場合には、環状メモリ・アレイ167に、ユーザ・タスク179によって消費されなければならないデータが含まれる。

【0060】オーナー・データ駆動プロトコルは、オーナー・タスク177が所与の速度でデータを製作しなければならないが、ユーザ・タスク179がその消費速度をオーナー・タスク177の速度に合わせることができる時に有用である。オーナー・データ駆動プロトコルを必要とする応用例が、電話回線からデータを受け取り、処理のため別のタスクにデータを渡すモデムである。電話回線からの着信データをサンプリングするタスクは、これを正確に固定された速度で行わなければならない。データを処理するタスクの時間要件は、これほど厳しくはないであろう。

【0061】図20は、ユーザ・タスク179がそれ自体の速度でデータを読み取るユーザ・データ駆動プロトコルを示す図である。オーナー・タスク177は、ユーザ・タスク179の速度に一致することが期待され、必ず環状メモリ・アレイ167内に十分なデータを保持しなければならない。ユーザ・タスク179は、一定速または可変速でデータを読み取ることができるが、この速度は、指定された最大速度以下でなければならない。オーナー・タスク177は、ユーザ・タスク179の最大データ消費速度（MWPF）に基づいて、書込ポインタ171と読取ポインタ173の間の間隔を維持する責任を負う。ユーザ・タスク179は、「開ループ」で走行し、書込ポインタ171のアドレスPUTPを知らない。対照的に、オーナー・タスク177は、ユーザ・タスク179の読取ポインタ173のアドレスAGETを知っている。

【0062】図21は、安全データ駆動プロトコル・モードの動作を示す図である。このプロトコルでは、オーナー・タスク177とユーザ・タスク179の両方が、積極的にポインタ・オーバーランを防止する。オーナー・タスク177は、書込ポインタ171と読取ポインタ173の位置を考慮して環状メモリ・アレイ167が「満杯」と判定される場合には、絶対に環状メモリ・アレイ167にデータを書き込まない。ユーザ・タスク179は、書込ポインタ171と読取ポインタ173の相対位置によって環状メモリ・アレイ167が「空」と判定される場合には、絶対に環状メモリ・アレイ167からデータを読み取らない。オーナー・タスク177が、1ブロックあたり16サンプルなどのブロック単位でユーザ・タスク179にデータを渡す場合、オーナー・タスク177は、書込動作を開始する前に、1ブロック分の空きが環状メモリ・アレイ167内に生じるまで待機しなければならない。ユーザ・タスク179は、読取動作を開始するために、1ブロック分のデータが使用可能になるまで待機しなければならない。この安全データ駆動プロトコルの動作では、オーナー・タスク177に、読取ポインタ173のアドレスが常に通知される。反対に、ユーザ・タスク179には、書込ポインタ171のアドレスが常に通知される。

【0063】データ通信モジュールへのすべての接続は、図44に示されるデータ通信モジュール・マクロに関して確立し、定義することができる。データ通信モジュールへの接続のそれぞれは、下記のラベルおよびパラメータの参照によって定義される。

（１）ラベル（label）
（２）オーナーまたはユーザの指定（OWNER、USER）
（３）サイズまたは最小サイズ（size、MINSIZE）
（４）データ通信プロトコル（PROTOCOL）
（５）データ読取またはデータ書込のいずれかの最大ワード毎秒（MAXWPF）
（６）最小ポインタ間隔（MPTRSEP）
（７）データ・サンプル速度（SAMPRATE）
（８）データ・アドレッシング・モード（ADDRMODE）
（９）データ・フォーマット・モード（MODE）
（１０）読取動作と書込動作の間の最大遅延（MAXDELT、MAXDELF）
（１１）そのデータ通信モジュールを通過するデータ要素のサイズ（STRIDE）

【００６４】これらのパラメータのそれぞれを、下でマクロを参照して詳細に説明する。マクロは、アセンブラ・プログラミング言語の形で図４４に示されている。

【００６５】LABEL FOR THE DATA COMMUNICATION MODULE（データ通信モジュールのラベル）：ラベルは、データ通信モジュールを指定するために必要である。これは、モジュラー・マルチメディア・ソフトウェア・タスク間またはモジュラー・マルチメディア・ソフトウェア・タスクとマルチメディア末端装置の間でデータを受け渡しするために作成された他のデータ通信モジュールから特定のデータ通信モジュールを識別し、区別するのに使用される。

【００６６】OWNER OR USER DESIGNATION（オーナーまたはユーザの指定）：データ通信モジュールへの接続を、オーナー・タイプまたはユーザ・タイプのいずれかとして識別することが必要である。両方を指定することはできない。

【００６７】SIZE（サイズ）：環状メモリ・アレイのサイズを、１６ビット・ワード単位で指定する。好ましい実施例では、データ通信モジュールが３２、６４、１２８、２５６、５１２、１０２４または２０４８ワードのサイズを有することができる。同期式プロトコル・モードの動作でのデータ通信モジュールの動作のためには、バッファのサイズが、製作または消費される最大ワード毎フレームの少なくとも２倍であることが重要である。代替案では、ユーザが、ユーザ・タイプのデータ通信モジュールのみに関して最小バッファ・サイズを選択できる。ユーザ・タイプのデータ通信モジュールは、最小サイズ（MINSIZE）以上のサイズを有するオーナー・タイプのデータ通信モジュールに接続することができる。

【００６８】PROTOCOL（プロトコル）：上で述べたように、同期式プロトコル、オーナー・データ駆動プロトコル、ユーザ・データ駆動プロトコルおよび安全データ駆動プロトコルを含む４つのプロトコルを、ユーザが選択することができる。図２２に、最下位５ビットがデータ通信プロトコルの選択を表す１６ビット・ワードを示す。ビット０の２進数１は、プロトコルが選択されていないことを示す。ビット１の２進数１は、同期式プロトコル・モードの動作がユーザによって選択されたことを示す。ビット２の２進数１は、オーナー・データ駆動プロトコルが選択されたことを示す。ビット３の２進数１は、ユーザがユーザ・データ駆動プロトコルを選択したことを示す。最後に、ビット４の２進数１は、安全データ駆動プロトコルが選択されたことを示す。ユーザは、プロトコルに互換性があるならば、複数のプロトコルを選択することができる。図４５に、プロトコルの互換性を示す。図からわかるように、同期式プロトコル・モードの動作で動作するオーナー・タイプのタスクは、データ通信モジュールを介して、同期式（SYNC）プロトコルモードの動作またはオーナー・データ駆動（OWNERDD）プロトコル・モードの動作のいずれかのユーザ・タイプのタスクと接続することができる。図４５には、さらに、オーナー・データ駆動プロトコル・モードの動作で動作するオーナー・タイプのタスクが、データ通信モジュールを介して、オーナー・データ駆動プロトコル・モードの動作のユーザ・タイプのタスクだけに結合できることが示されている。図４５には、さらに、ユーザ・データ駆動（USERDD）プロトコル・モードの動作で動作するオーナー・タイプのタスクが、データ通信モジュールを介して、識別される４つのプロトコル・モードの動作のどれで動作するユーザ・タイプのタスクにでもデータを渡せることが示されている。最後に、図４５には、安全データ駆動（SAFEDD）プロトコル・モードの動作で動作するオーナー・タイプのタスクが、オーナー・データ駆動プロトコル・モードの動作または安全データ駆動プロトコル・モードの動作のいずれかで動作するユーザ・タイプのタスクにデータを渡せることが示されている。

【００６９】MAXIMUM WORDS PER FRAME（最大ワード毎フレーム）：最大ワード毎フレームは、タスク実行のたびに製作または消費されるワードの最大数である。たとえば、オーナー・タスクが毎フレーム１６ワードの一定速度でデータを書き込む場合、最大ワード毎フレーム（MAXWPF）は１６である。

【００７０】MINIMUM SEPARATION（MPTRSEP、最小間隔）：最小間隔は、オーナー・ポインタとユーザ・ポインタの間で許容することのできる最少の間隔である。同期式プロトコル・モードの動作で動作しているデータ通信モジュールの場合、経験則として、最小間隔を最大ワード毎フレームにセットするのがよい。オーナー・タスクとユーザ・タスクが異なる値の最小間隔を定義する場合、大きい方の値を使用して、これら２つの動作を同期化させなければならない。

【００７１】SAMPLE RATE（SAMPRATE、サンプル速度）：サンプル速度は、データ通信モジュールを通って流れるデータの固有のサンプル速度を指定するパラメータである。ユーザとオーナーの両方が０以外のサンプル

速度を指定する場合、サンプル速度は、オーナー・タスクとユーザ・タスクの両方について同一でなければならない。そうでない場合には、それらのタスクを接続することができない。

【００７２】DATA ADDRESSING MODE（ADDRMODE、データ・アドレッシング・モード）：データ・アドレッシング・モードは、データ通信モジュールのポインタを操作するのに使用されるモードを識別するパラメータである。選択肢は、"byte（バイト）"、"table（テーブル）"または"word（ワード）"である。オーナー・タスクとユーザ・タスクを接続するためには、これらが同一のタイプのデータ・アドレッシング・モードを識別しなければならない。

【００７３】DATA FORMAT MODE（MODE、データ・フォーマット・モード）：データ・フォーマット・モードは、データ通信モジュールに書き込まれ、読み取られるデータのフォーマットを記述するパラメータである。オーナー・タスクとユーザ・タスクは、同一のモードを指定しなければならない。そうでない場合、接続が許可されない。データは、長さN（Nは整数）のバケットとみなされる。たとえば、Nを2にセットするという技法は、ステレオ・サウンドの右チャンネルと左チャンネルの伝送や、複素数の実数部と虚数部を伝送を可能にする技法の1つである。Nを他の整数値にセットすると、ある信号の異なる部分を表すN個の構成要素内のより大きなデータの集合を受け渡しできるようになる。データ・フォーマット・モードを"universal（汎用）"にセットすると、データ・フォーマットに関する具体的なモード選択に無関係に、データ通信モジュールを別のデータ通信モジュールに接続できるようになる。

【００７４】MAXIMUM DELAY（MAXDELT,MAXDELF、最大遅延）：最大遅延は、オーナー・タスクがデータ通信モジュールにデータを書き込む時刻とユーザ・タスクがデータ通信モジュールからデータを読み取る時刻の間で許容される時間を表すパラメータである。最大遅延は、"MAXDELF"または"MAXDELT"のいずれかとして指定できるが、両方を指定することはできない。MAXDELTは、ナノ秒、マイクロ秒、ミリ秒または秒単位の絶対時間で測定された時間である。MAXDELFの測定単位は、データ通信モジュール・マクロに関連するタスクの1フレームまたは1周期である。たとえば、あるオーナー・タスクが8KHzの割込み供給源からの割込みの8回に1回走行する場合、そのタスクのフレームまたは周期は、1ミリ秒と判定される。このタスクは、それが通信モジュールにデータを書き込んだ時刻とそのデータが消費される時刻の間に3ミリ秒以内の遅延しか許容できない。このデータ通信モジュール・マクロでは、MAXDELT=3msまたはMAXDELF=3のいずれかによって遅延許容範囲を指定しなければならない。

【００７５】STRIDE（ストライド）：ストライドは、データ通信モジュールを通過するデータ要素の、ワード単位のサイズである。ストライドは、タスクに有用な1単位の情報を含む16ビット・サンプルの個数を定義するパラメータである。N次元データを処理している場合、ストライドは、Nにセットしなければならない。たとえば、ステレオ・オーディオは、左右のサンプル対の形で処理しなければならないので、ストライドを2にセットしなければならない。

【００７６】図23は、図44に示された、データ通信モジュールを定義するマクロで行われた指定の結果としての、オーナー・タスクとユーザ・タスクでの制御ブロックの作成を示す図である。製作側であるオーナー・タスク177を環状メモリ・アレイ167に接続し、消費側であるユーザ・タスク179を環状メモリ・アレイ167に接続するために、2つのマクロ（図44に示されたものなど）を定義しなければならない。オーナー・タイプのマクロによって、オーナー・タスクのデータ・セグメントに制御ブロック181が作成される。ユーザ・タイプのマクロによって、ユーザ・タスクのデータ・セグメントに制御ブロック183が作成される。制御ブロックの長さは、選択されたデータ通信プロトコルに応じて、1ワードから4ワードまでの間で変化する。生成される可能性のある異なるタイプの制御ブロックを、図18ないし図21に示す。

【００７７】図18に戻って、オーナー・タスク177内に確立された制御ブロックには、書込ポインタ171だけが含まれる。ユーザ・タスク179には、読取ポインタ173（GETP）だけが含まれる。図19では、オーナー・タスク177に、書込ポインタ171（PUTP）だけが含まれる。ユーザ・タスク179では、制御ブロックに、読取ポインタ173（GETP）、環状メモリ・アレイのサイズ（SIZE）、最大ワード毎フレーム（MWPF）および書込ポインタ171のアドレス（APUT）が含まれる。図20では、オーナー・タスク177が、書込ポインタ171（PUTP）、環状メモリ・アレイのサイズ（SIZE）、最大ワード毎秒（MWPF）および読取ポインタ173のアドレス（AGET）を含む制御ブロックを有する。ユーザ・タスク179には、読取ポインタ173だけが含まれる。図21では、オーナー・タスク177の制御ブロックに、書込ポインタ171のアドレス（PUTP）と読取ポインタ173のアドレス（AGET）の両方が含まれる。ユーザ・タスク179にも同様に、読取ポインタ173(GETP)と書込ポインタ171のアドレス（APUT）が含まれる。

【００７８】データ通信モジュール用の堅固なデータ通信構造と標準化データ通信プロトコルを有することの実質的な利益の1つが、データ通信モジュールに対する読取動作と書込動作を、単純な標準化されたソフトウェア・ルーチンとして構築できることである。図28は、同期式プロトコル・モードの動作でデータ通信モジュール

にデータを書き込むタスクの流れ図である。図２９は、同期式プロトコル・モードの動作でデータ通信モジュールからデータを読み取るタスクの流れ図である。

【００７９】まず図２８を参照すると、書込動作は、ブロック２０１から始まる。ブロック２０３で、ディジタル信号プロセッサに、オーナー（書込）ポインタをゲットするように命令する。ディジタル信号プロセッサは、インデクシングを可能にするためにブロック２０５で停止する。この実施例の特定のDSPは、アドレス・レジスタの更新に１クロック・サイクルを必要とする。ブロック２０７で、ディジタル信号プロセッサが、選択されたデータ通信モジュールを構成する環状バッファにデータを書き込む。その後、ブロック２０９で、ディジタル信号プロセッサが、オーナー（書込）ポインタを増分して、環状バッファ内の次のメモリ・セグメントへの書込を可能にする。書込動作が完了するまで、ブロック２０７とブロック２０９を繰り返す。ブロック２１１で、ディジタル信号プロセッサは、将来の書込動作で使用するためにポインタの値をセーブする。ブロック２１３で、このルーチンは、DSPメモリ内に常駐するDSPオペレーティング・システムに戻る。ブロック２１５で、書込処理が停止する。

【００８０】データをデータ通信モジュールから読み取る時の処理は、そのタスクがDSPオペレーティング・システムによって処理のためにスケジューリングされた時に、ブロック２１７から始まる。ブロック２１９で、ディジタル信号プロセッサに、ユーザ（読取）ポインタをゲットするように命令する。ブロック２２１で、ディジタル信号プロセッサがインデクシング動作を待つ。ブロック２２３で、ディジタル信号プロセッサが、環状バッファからデータを読み取る。次に、ブロック２２５で、ディジタル信号プロセッサが、ユーザ（読取）ポインタを増分する。読取動作が完了するまで、ブロック２２３とブロック２２５を繰り返す。ブロック２２７で、ディジタル信号プロセッサが、将来の読取動作に使用するためにユーザ（読取）ポインタの値をセーブする。その後、ブロック２２９で、このルーチンはDSPメモリに常駐するDSPオペレーティング・システムに戻る。

【００８１】図３０は、ユーザ・データ駆動プロトコル・モードの動作でデータ通信モジュールにデータを書き込むタスクの流れ図である。この処理は、DSPオペレーティング・システムによってタスクが実行のためにスケジューリングされた時に、ブロック２３５から始まる。ブロック２３７で、ディジタル信号プロセッサに、オーナー（書込）ポインタの値を判定するよう命令する。次に、ブロック２３９で、ディジタル信号プロセッサに、ユーザ（読取）ポインタの値を判定するよう命令する。その後、ブロック２４１で、ディジタル信号プロセッサに、オーナー（書込）ポインタとユーザ（読取）ポインタの間の距離を判定するよう命令する。ブロック２４３

で、ディジタル信号プロセッサに、判定されたオーナー（書込）ポインタとユーザ（読取）ポインタの間の距離に応じた速度で、データ通信モジュールにデータを書き込むように命令する。ブロック２４５で、ディジタル信号プロセッサに、オーナー（書込）ポインタとユーザ（読取）ポインタの両方のポインタの値をセーブするよう命令する。ブロック２４７によって、このルーチンは、DSPメモリに常駐するDSPオペレーティング・システムに戻る。最後に、ブロック２４９で、処理が停止する。

【００８２】図３１は、オーナー・データ駆動プロトコル・モードの動作または安全データ駆動プロトコル・モードの動作のいずれかでデータ通信モジュールからデータを読み取るタスクを示す流れ図である。この処理は、DSPオペレーティング・システムが実行のためこのルーチンをスケジューリングした時に、ブロック２５１から始まる。ブロック２５３で、ディジタル信号プロセッサに、オーナー（書込）ポインタの値を判定するよう命令する。次に、ブロック２５５で、ディジタル信号プロセッサに、ユーザ（読取）ポインタの値を判定するよう命令する。ブロック２５７で、ディジタル信号プロセッサに、オーナー（書込）ポインタとユーザ（読取）ポインタの間の距離を判定するよう命令する。次に、ブロック２５９で、ディジタル信号プロセッサに、ポインタ位置から環状メモリ・アレイが「空」であることが示されるまで、データ通信モジュールから使用可能データを読み取るよう命令する。次に、ブロック２６１で、ディジタル信号プロセッサに、将来の読取動作で使用するためにポインタの値をセーブするよう命令する。ブロック２６３で、ディジタル信号プロセッサが、DSPメモリに常駐するDSPオペレーティング・システムに制御を返す。最後に、ステップ２６５で、この処理が終了する。

【００８３】上で述べたように、単一のユーザ・タスクが、複数のデータ通信モジュールを介してストリーム化されたデータを受け取ることが可能である。図２４は、複数のモジュラー・マルチメディア・ソフトウェア・タスク２６７、２６９、２７１および２７３から、複数のデータ通信モジュール２７５、２７７、２７９および２８１を介して単一のユーザ・タスク２９３への、仮想データ通信モジュール２９１の使用を介するデータ・ストリームの通信を示すブロック図である。本発明の好ましい実施例では、複数の「ミュート」タスク２８３、２８５、２８７および２８９が、データ通信モジュール２７５、２７７、２７９および２８１と仮想データ通信モジュール２９１の間に接続される。ミュート・タスク２８３、２８５、２８７および２８９は、データ通信モジュール２７５、２７７、２７９および２８１の出力に減衰係数または「ミュート」係数を適用して、出力を減衰させ、仮想データ通信モジュール２９１によって実行される合計演算中のオーバーフローを防ぐ。

【００８４】この種の動作の例を、図２６に示す。図２６は、シンセサイザ・タスク２９５、モデム・オーディオ・タスク２９７およびアラーム・クロック・サウンド・タスク２９９を含む複数のモジュラー・マルチメディア・ソフトウェア・タスクから単一のCDディジタル・オーディオ（CD-D/A）変換器タスク３０７への、オーディオ出力信号の多重化を示すブロック図である。リアルタイム・データまたは非同期ストリーム化データが、シンセサイザ・タスク２９５からデータ通信モジュール３０１を介して仮想データ通信モジュール３１７に向けられる。同様に、リアルタイム・データまたは非同期ストリーム・データが、モデム・オーディオ・タスク２９７からデータ通信モジュール３０３を介して仮想データ通信モジュール３１７に向けられる。同様に、アラーム・クロック・サウンド・タスク２９９は、リアルタイム・データまたは非同期ストリーム化データを、データ通信モジュール３０５を介して仮想データ通信モジュール３１７に向ける。仮想データ通信モジュール３１７は、シンセサイザ・タスク２９５、モデム・オーディオ・タスク２９７およびアラーム・クロック・サウンド・タスク２９９からのさまざまなリアルタイム・データまたは非同期ストリーム化データ出力を合計して、CD-D/Aタスク３０７への入力を提供する。CD-D/Aタスク３０７は、仮想データ通信モジュール３１７からのディジタル出力を、ディジタル化アナログ出力に変換する。この出力は、データ通信モジュール３０９および３１１を介して渡された後に、スピーカ３１３および３１５に向けられ、オーディオ・サウンド出力をもたらす。

【００８５】図２６に示された例では、シンセサイザ・タスク２９５からの出力が、合成された人間の音声を表す可能性がある。モデム・オーディオ・タスク２９７からの出力は、モデム・タスクが実行中であることをユーザに示すために、市販の大半のモデムで実現されている聴取可能な確認トーンを表す可能性がある。アラーム・クロック・サウンド・タスク２９９は、所定の時刻に予定された事象についてユーザに警告するためのアラーム・クロック・サウンドを表す可能性がある。仮想データ通信モジュール３１７を使用すると、これらのタスクからのリアルタイム・データまたは非同期ストリーム化データを用いてスピーカ３１３および３１５を同時に駆動できるようになる。これは、マルチメディア末端装置を駆動するさまざまなマルチメディアタスクのオーバーラップする同時動作が、特定のタスクや末端装置のいずれの動作にも干渉せずに可能になるので、有利な特徴である。この形で、図２６は、定義済みの標準化デー通信プロトコルおよびデータ通信規則と共に使用される本発明のデータ通信モジュールの実世界での利点の１つを示すものである。

【００８６】図２５は、図２４および図２６の仮想データ通信モジュールの動作を示す詳細なブロック図である。図２５からわかるように、オーナー・タスク３１９は、書込ポインタ３３１を使用してデータ通信モジュール３２５にデータを書き込み、オーナー・タスク３２１は、書込ポインタ３３３を使用してデータ通信モジュール３２７にデータを書き込み、オーナー・タスク３２３は、書込ポインタ３３５を使用してデータ通信モジュール３２９にデータを書き込む。読取ポインタ３３７、３３９および３４１によって、ユーザ・タスク３４３、データ・マクロ３４５、３４７および３４９の命令とパラメータに従い、データ通信モジュール３２５、３２７および３２９からデータを読み取る。図からわかるように、データ・マクロ３４５、３４７および３４９は、図２５の右側に描かれた仮想データ通信モジュール制御ブロックを介して、リンク・リストによって互いに結合される。図からわかるように、リンク３５１によってユーザ・タスク３４３がデータ・マクロ３４５に接続される。リンク３５３によって、データ・マクロ３４５がデータ・マクロ３４７に接続される。リンク３５５によって、データ・マクロ３４７がデータ・マクロ３４９に接続される。

【００８７】ユーザ・タスク３４３には、データ・マクロ３４５へのアドレス（VDCMマクロ）が含まれる。データ・マクロ３４５によって、ミュート係数、読取ポインタ（GETP）、データ通信モジュール３２５のサイズ（SIZE）、データ通信モジュール３２５への最大データ・フロー速度（最大ワード毎フレーム、MWPF）および書込ポインタ３３１のアドレス（APUT）が定義される。データ・マクロ３４５には、さらに、オーナー・タスク３１９からのデータを処理する際にユーザ・タスク３４３によって使用することのできる（しなくてもよい）ユーザ定義変数が含まれる。データ・マクロ３４５には、次の仮想データ通信モジュールへのアドレスが含まれる。図からわかるように、データ・マクロ３４７には、読取ポインタ３３９（GETP）、データ通信モジュール３２７のサイズ（SIZE）、データ通信モジュール３２７への最大データ・フロー速度（最大ワード毎フレーム、MWPF）および書込ポインタ３３３のアドレス（APUT）が含まれる。データ・マクロ３４７には、さらに、オーナー・タスク３２１からのデータを処理する際にユーザ・タスク３４３によって使用することのできるユーザ定義変数が含まれる。データ・マクロ３４７には、次の仮想データ通信モジュールへのアドレスも含まれる。図からわかるように、データ・マクロ３４９には、読取ポインタ３４１の値（GETP）が含まれる。データ・マクロ３４９は、最後のデータ多重化タスクを表すので、その最初の位置を"0000"にセットして、制御を他のデータ・マクロに渡してはならないことを示す。

【００８８】図４６は、仮想データ通信マクロの基本概念を表の形で識別する図である。最初の構成要素は、「次の仮想DCM」として識別される、「次の仮想データ

通信モジュール」へのアドレスである。第２の構成要素は、接続中に出力を静かにするのに使用されるミュート係数（Mute)である。第３の構成要素は、ユーザのポインタ、環状メモリ・バッファのサイズ、最大ワード毎フレームおよびオーナーのポインタのアドレスを識別する制御ブロックである。最後に、他の変数のためにユーザ定義データを設ける。

【００８９】上で述べたように、データ通信モジュールは、これまで完全には説明してこなかったタスク間制御ブロック（ITCB）と共同して働く。図２７は、タスク間制御ブロックの動作を示すブロック図である。各タスクは、ディジタル信号オペレーティング・システムによって実行のために呼び出されるが、他のタスクからは不可視のそれ自体の局所データ・セクションを有する。タスク間制御ブロックは、あるタスクが別のタスクと１ブロックのデータを共用できるようにする機構である。タスク間制御ブロック（ITCB）を介して通信するタスクを、「主タスク」および「副タスク」と呼ぶ。図２７に、主タスク３５７、副タスク３５９およびその相互接続を示す。図からわかるように、ホストCPUに常駐するホスト・アプリケーション３６１は、アプリケーション・プログラム・インターフェース（API）コマンド"connect ITCB（ITCB接続）"と"disconnect ITCB（ITCB切断）"を使用して、ポインタ３６７の接続と切断を行う。このポインタは、副タスク３５９のデータ・セグメントから、主タスク３５７のデータ・セグメント部分にある制御ブロック３６９を指す。したがって、選択されたモジュラー・マルチメディア・ソフトウェア・タスク間で状況と制御変数を受け渡しする制御ブロック３６９は、主タスクのデータ・セクションに置かれる。主タスク３５７には、データの構造を定義するマクロが含まれる。副タスク３５９には、制御ブロック３６９の先頭を差すポインタ３６７を宣言するマクロが含まれる。

【００９０】前に述べたように、ホスト・アプリケーション３６１は、定義済みのディジタル信号プロセッサ・コマンドを用いて副タスク３５９を主タスク３５７に接続する。タスクが接続される時に、図３のDSPマネージャ７１が、副タスク３５９のデータ・セクション内の制御ブロック３６９を指すポインタを記憶する。ホスト・アプリケーション３６１は、いつでも主タスク３５７から副タスク３５９を切断したり接続することができる。タスクが接続された時、副タスク３５９は、主タスク３５７の制御ブロック３６９のデータを読み書きすることができる。副タスク３５９は通常、制御ブロック３６９のベースを指すポインタ３６７をインデックス・レジスタにロードし、制御ブロック３６９内のデータをベースからのオフセットとしてアドレッシングする。

【００９１】タスク間制御ブロックの作成と使用に必要な全体的なシーケンスをまとめると、次のようになる。

（１）タスク間制御ブロックを、（１）ITCB PRIMARY（主ITCB）、（２）ITCBLBLおよび（３）EITCBを含む３つのマクロを使用して、主タスクのデータ・セクション内でコード化する。これら３つのマクロは、下で説明する。

（２）副ITCBを、副タスクのデータ・セクション内にマクロを介してコード化する。このマクロは、主タスクのITCBデータ・ブロックの先頭を指すポインタのために１ワードのメモリを割り振る。

（３）実行時に、ホスト・システムのアプリケーションが、DSPマネージャへの呼出しをセットすることによって、副タスクを主タスクに接続する。その後、DSPマネージャが、副タスクのポインタに、タスク間制御ブロックのベース・アドレスをロードする。もちろん、ホスト・アプリケーションは、いつでも副タスクを接続したり切断することができる。

（４）タスクが接続された後には、副タスクがタスク間制御データ・ブロックを読み書きできる。通常、副タスクは、このデータ・ブロックのベースを指すポインタをインデックス・レジスタにロードし、ベースからのオフセットとしてこのレジスタ内でデータをアドレッシングする。副ポインタは、接続が存在しない時には０にセットされる。

【００９２】主タスクのデータ・セクション内のITCB PRIMARYマクロは、ITCBデータ・ブロックの先頭を識別し、EITCBマクロは、制御データ・ブロックの末尾を識別する。ITCBLBLマクロは、所与のITCB制御データ・ブロック内の単一の変数または変数のグループの属性を定義する。このマクロによって、下記の情報が定義される。

（１）読取動作と書込動作に関して、変数のために確立されたITCBプロトコル

（２）あるタスクから別のタスクに変数を渡す際に許容される最大遅延

（３）１つまたは複数の変数のラベル

【００９３】タスク間制御ブロックの機能ならびに、データ通信モジュールの全体動作および長所は、図３２、図３３および図３４に示された例を参照することによって最もよく理解できる。図３２は、複数のモジュラー・マルチメディア・ソフトウェア・タスクの動作を示すブロック図であり、これらのタスクの一部は、通常のハードウェア・マルチメディア末端装置の動作を仮想化する。図３３は、オーディオ出力増幅タスクの動作を示すブロック図である。図３４は、データ・ストリーム間の位相差を制御し操作するためのデータ通信モジュールとモジュラー・マルチメディア・ソフトウェア・タスクの使用を示すブロック図である。

【００９４】まず図３２を参照すると、全二重モードで動作するモデムの機能を実行するよう構成された、本発明のマルチメディア・データ処理システムの動作が示されている。CPU３３が、ディジタル信号プロセッサ３７

にデータを送り、これからデータを受け取る。具体的に言うと、データは、汎用非同期送受信器（UART）レジスタ３７１を介して渡される。ストリーム化データは、ディジタル信号プロセッサ３７からデータ通信モジュール３７３を介して送られる。このデータ通信モジュールは、モデム制御タスク３７７へのデータの連続的なリアルタイム単一方向通信を可能にする。データ通信モジュール３７５は、連続的なリアルタイム単一方向データを、モデム制御タスク３７７からディジタル信号プロセッサ３７に向けるように働く。データ通信モジュール３７９は、ストリーム化データをモデム制御タスク３７７からデータ変換タスク３８３に向けるように働く。データ通信モジュール３８１は、データをデータ変換タスク３８３からモデム制御タスク３７７に向けるように働く。データ通信モジュール３８５は、データをデータ変換タスク３８３から電話タスク３８９に向けるように働く。データ通信モジュール３８７は、データを電話タスク３８９からデータ変換タスク３８３に向けるように働く。電話タスク３８９は、データ通信モジュール３９１を介して電話入出力回線と通信する。図からわかるように、データ通信モジュールは、全二重モードで動作するモデムのデータ・フロー経路に対応する２つのデータ・フロー経路をもたらす。

【００９５】モデム制御タスク３７７は、電話入出力からデータ通信モジュール３９１で受け取られる可能性があるRING（呼出）信号を含むさまざまな変数に反応するソフトウェア状態機械を表す。タスク間通信ブロック３９５は、モデム制御タスク３７７をデータ通信モジュール３９１に連結し、電話入出力回線上でRINGを受け取ったことをモデム制御タスク３７７に警告するために、データ通信モジュール３９１からモデム制御タスク３７７へ変数RINGを渡せるようにする。

【００９６】データ変換タスク３８３は、モデム制御タスク３７７からディジタル・データ・ストリームを受け取り、そのディジタル・データ・ストリームに対応するディジタル化アナログ信号を表す出力ストリームを作るように動作する。タスク間制御ブロック３９３は、データ変換タスク３８３とモデム制御タスク３７７の間に結合され、これらのタスクの間で変数RATE（速度）を通信するように働く。RATEは、データ変換タスク３８３からディジタル化アナログ形式でデータを出力しなければならない速度を識別する変数である。

【００９７】図３２の例によれば、状況標識RINGと制御変数RATEをこれらのタスクとデータ通信モジュールの間で渡して、モジュラー・マルチメディア・ソフトウェア・タスクの調整された動作を保証する。図３２からわかるように、通常の電話の機能を、モジュラー・マルチメディア・ソフトウェア・タスクである電話タスク３８９によって、ソフトウェア的にシミュレートすることができる。同様に、モデム制御タスク３７７によって、通常のモデムの通常のハードウェアおよびソフトウェアの機能をシミュレートする。したがって、本発明は、さまざまな従来のハードウェア末端装置を「仮想化」するのに有用であり、したがって、ユーザに、既存の従来技術システムから拡張された柔軟性を提供することが明白である。ユーザは、通常は従来のマルチメディア末端装置によって実行されるはずのタスクを、CPUおよびディジタル信号プロセッサによって実行されるようにコード化することができる。ハードウェア装置の従来の機能を実行するソフトウェア・ルーチンによってハードウェア装置を「仮想化」することによって、かなりの節約が可能になる。この技法を使用してさまざまなマルチメディア末端装置を仮想化することによって、エンド・ユーザに、マルチメディア環境に対する拡張された制御が与えられる。

【００９８】図３３は、タスクが制御変数を読み書きできるかどうかを判定するのに使用されるITCBプロトコルの使用例を示す図である。図からわかるように、データ通信モジュール３９７は、ストリーム化データを増幅および電力計算タスク３９９に通信する。データ通信モジュール４０１は、増幅および電力計算タスク３９９の出力を受け取り、これをステレオ出力に向ける。タスク間制御ブロック（ITCB）４０３は、別の（非開示の）タスクから増幅および電力計算タスク３９９へ変数を通信する。

【００９９】ITCB４０３には、２つの変数すなわち、"GAIN L"および"GAIN R"として識別される左利得４０５と右利得４０７が含まれる。図では、ITCBLBLマクロが、破線によってITCB４０３に接続されている。このマクロは、このタスク間制御ブロックの名前を"AMPGAIN"として識別する。このマクロは、第１の変数を"GAIN L"として識別し、この変数の読書性を"RW"として識別する。ITCBLBLマクロは、第２の変数を"GAIN R"として識別し、この変数の読書プロトコルを"RW"として識別する。さらに、ITCBLBLマクロは、第３の変数を"POWER（電力）"として識別する。この変数は、"WR"という読書アクセス・プロトコルを有する。

【０１００】GAIN LおよびGAIN Rの"RW"というアクセス・プロトコルによって、主タスク（増幅および電力計算タスク３９９）が、ITCB４０３から変数GAIN LおよびGAIN Rを読み取れることが識別される。このアクセス・プロトコルによって、副タスク（非開示のタスク）が、変数GAIN LおよびGAIN Rの位置に変数を書き込み可能であることも識別される。この形で、ユーザは、マルチメディア・ソフトウェア・アプリケーション制御タスクを有することができ、このタスクは、利得を修正するために"GAIN L"と"GAIN R"に変数を書き込むことができる。これらの変数は、増幅および電力計算タスク３９９によって使用され、データ通信モジュール３９７によって提供されるデータ・ストリームの増幅をもたらす。たとえ

ば、GAIN LとGAIN Rを2にセットする場合、増幅および電力計算タスク399の受け取るデータの値が、2倍に増幅される。副タスク（すなわち、非開示のタスク）は、変数GAIN LおよびGAIN Rに変数を書き込むことができ、したがって、これらをより高い値またはより低い値に修正して、増幅および電力計算タスク399の利得を修正することができる。

【0101】その名前が暗示するとおり、増幅および電力計算タスク399は、データ通信モジュール401に供給される信号の電力出力を計算する。この電力変数411は、周期的に変数"POWER"に書き込むことによって、副タスクに通信することができる。電力値が変化する際に、その変更が変数"POWER"の修正によって反映される。この例によって示されるように、ITCBLBLマクロによって、読み書きアクセス・プロトコルが決定され、このプロトコルによって、主タスクと副タスクのどちらが変数に書き込めるかと、主タスクと副タスクのどちらが変数から読み取れるかが決定される。

【0102】図34に示されるように、本発明のデータ通信モジュールとタスク間制御ブロックを使用して、データのストリームの位相特性を変更することができる。図からわかるように、データ通信モジュール413は、ストリーム化データを位相遅延タスク415に向け、位相遅延タスク415は、所定の量と可変量の位相遅れを導入する。位相遅延タスク415は、ストリーム化データ出力を作り、このストリーム化データ出力は、データ通信モジュール417に向けられ、データ通信モジュール417は、直接または間接的に、ステレオ出力にストリーム化データ出力を供給する。タスク間制御ブロック（ITCB）419によって、位相遅延タスク415をもう1つの（非開示の）タスクに結合する。"PHASE（位相）"変数421が、ITCB419によって、具体的にはITCBLBLマクロ423によって定義され、このITCBLBLマクロ423によって、ITCB419に"PHASEDELAY（位相遅延）"という名前を付け、"RW"としてアクセス・プロトコルを定義する。このプロトコルによれば、主タスクは、"PHASE"変数421の値を読み取ることができ、副タスクは、"PHASE"変数421に値を書き込むことができる。位相遅延タスク415が「主」タスクの場合、"PHASE"変数421の値を読み取ることはできるが、それに書き込むことはできない。この形で、ユーザが、モジュラー・マルチメディア・ソフトウェア・タスクまたはマルチメディア末端装置からの特定のデータストリーム出力のどれに対しても、選択された量の位相遅れを導入したり除去することができるマルチメディア・アプリケーションを構築できる。この動作は、位相遅延タスクの知的使用を介してマルチメディア・アプリケーションの厄介な位相遅れを除去するという利点を有する。したがって、同期化の問題も、完全に回避することができる。

【0103】図3からわかるように、DSPマネージャ71は、ホストCPUに常駐し、ディジタル信号プロセッサ37（図2）へのモジュラー・マルチメディア・ソフトウェア・タスクの実行のためのロードを制御するように動作する。図35からわかるように、DSPマネージャ71は、互いに協力してモジュラー・マルチメディア・ソフトウェア・タスクのロードと実行を制御する複数の機能ブロックからなる。互いに協力するブロックには、ディジタル信号プロセッサ（DSP）パーサ451、ディジタル信号プロセッサ（DSP）リンク・エディタ453、ディジタル信号プロセッサ（DSP）資源マネージャ455、タスク・ローダ457、タスク・マネージャ459、ディジタル信号プロセッサ・マネージャ・アプリケーション・プログラム・インターフェース（DSPMGR API）コントローラ461、割込みおよびプロセッサ・コール・バック（IPC）ハンドラ463およびディジタル信号プロセッサBIOS（DSP BIOS）インターフェース465が含まれる。これらの機能ブロックのそれぞれを、図36ないし図43に詳細に示す。

【0104】図35を参照すると、DSPMGR APIコントローラ461は、マルチメディア・アプリケーション59からアプリケーション・プログラム・インターフェース（API）コマンドを受け取り、そのアプリケーション・プログラム・インターフェース（API）コマンドを適切な機能ブロックに経路指定する。アプリケーション・プログラム・インターフェース・コマンドには、大別して5つのカテゴリーがある。カテゴリー1には、DSPMGR APIコントローラ461とタスク・ローダ457の間に向けられるコマンドが含まれる。APIコマンドのカテゴリー2には、DSPMGR APIコントローラ461からタスク・マネージャ459に渡されるコマンドが含まれる。APIコマンドのカテゴリー3には、DSPMGR APIコントローラ461からDSP BIOSインターフェース465に渡されるコマンドが含まれる。APIコマンドのカテゴリー4には、DSPMGR APIコントローラ461とIPCハンドラ463の間で渡されるコマンドが含まれる。APIコマンドのカテゴリー5には、状況その他の情報をDSPマネージャ71に提供する、ハウスキーピングやデバッグ動作に特に有用であるが、本発明を完全に理解するためには特に重要ではないコマンドが含まれる。

【0105】図47は、ホストCPUに常駐するDSPマネージャ71がディジタル信号プロセッサ37の動作を調整し制御できるようにするAPIコマンドの5つの大まかなカテゴリーを識別する表である。図47を参照すると、カテゴリー1のコマンドには、dsp allocate seg（DSPセグメント割振り）、dsp initiation（DSP開始）、dsp load module（DSPモジュール・ロード）、dsp load task（DSPタスク・ロード）およびdsp load thread（DSPスレッド・ロード）が含まれる。dsp allocate segは、ディジタル信号プロセッサに特定のタスクのためにメモリを確保するよう命令するコマンドである。dsp initia

tionは、レジスタをクリアし、プロセッサをリセットし、オペレーティング・システムをロードすることによって、ディジタル信号プロセッサを初期設定するコマンドである。dsp load moduleは、選択されたタスクのモジュール（図１０参照）をロードするコマンドである。dsp load taskは、ディジタル信号プロセッサに、単一のタスクを構成するデータ・セグメントと命令セグメントをロードするよう命令するコマンドである。dsp load threadは、実行のために単一のスレッド（すなわち、命令またはデータもしくはその両方の小さなブロック）をロードするよう命令するコマンドである。図３５からわかるように、カテゴリー１のコマンドは、DSPMGR APIコントローラ４６１からタスク・ローダ４５７に渡される。その詳細を図４３に示す。

【０１０６】図４３からわかるように、タスク・ローダ４５７には、制御ファイル４７３が含まれ、この制御ファイル４７３が、APIを受け取り、必要に応じてさまざまな他のファイルを呼び出す。モジュール・ロード・ファイル４６７は、タスクのモジュールをディジタル信号プロセッサにロードするタスクを実行する。タスク・ロード・ファイル４６９は、実行のため特定のタスクをディジタル信号プロセッサにロードする。スレッド・ロード・ファイル４７１は、実行のため特定のスレッドをディジタル信号プロセッサにロードする。

【０１０７】セグメント割振りファイル４７５は、特定のタスクの受け取りのためにメモリの特定の部分を割り振る。マネージャ初期設定ファイル４７７は、ディジタル信号プロセッサを選択的に初期設定する。モジュール・ロード・ファイル４６７、タスク・ロード・ファイル４６９およびスレッド・ロード・ファイル４７１は、DSPパーサ４５１、DSPリンク・エディタ４５３およびDSP資源マネージャと通信し、協力する。これらのすべてを、下で説明する。データ・イメージ・ロード・ブロック４７９、コード・イメージ・ロード・ブロック４８１および制御ブロック４８３は、互いに協力して、タスク・マネージャ４５９（図４１）に、ディジタル信号プロセッサへのロードのためにDSP BIOSインターフェース４６５を介して渡さなければならないデータ、命令および制御パラメータを供給する。

【０１０８】図４７からわかるように、APIコマンドのカテゴリー２には、１１個の異なるコマンドが含まれる。これらのコマンドのすべてが、DSPMGR APIコントローラ４６１からタスク・マネージャ４５９に通信される。dsp change CPF（DSP CPF変更）コマンドを使用すると、特定のタスクに割り振られるディジタル信号プロセッサの毎秒サイクル数（CPF）を変更できる。dsp change DMA（DSP DMA変更）コマンドを使用すると、直接メモリ・アクセス動作のために割り振られるメモリ資源の量を変更できる。DSP change module state（DSPモジュール状態変更）コマンドを使用すると、複数のマルチメディア・ソフトウェア・タスクのモジュラー・グループ化の状態を変更できる。使用可能な状態には、"active（活動）"、"standby（待機）"および"inactive（非活動）"が含まれる。"active"タスクとは、ロード時に即座にディジタル信号プロセッサによって実行されるタスクである。"standby"タスクとは、ディジタル信号プロセッサへのロードの後に、待ち行列化され、しばらく実行を待つタスクである。"inactive"タスクとは、"standby"状態でも"active"状態でもないタスクである。dsp change task（DSPタスク変更）コマンドを使用すると、単一のマルチメディア・ソフトウェア・タスクの状態を"active"、"standby"および"inactive"状態の間で変更できる。

【０１０９】dsp connect DCM（DSP DCM接続）コマンドを使用すると、特定のデータ通信モジュールを特定のタスクまたは特定のマルチメディア末端装置もしくはその両方に接続できる。dsp connect ITCB（DSP ITCB接続）コマンドを使用すると、主タスクと副タスクの間でタスク間制御ブロックを接続できる。dsp disconnect DCM（DSP DCM切断）コマンドを使用すると、データ通信モジュールを切断でき、dspdisconnect ITCB（DSP ITCB切断）コマンドを使用すると、タスク間制御ブロックを切断できる。dsp free module（DSPモジュール解放）、dsp free task（DSPタスク解放）およびdsp free thread（DSPスレッド解放）コマンドは、それぞれ、タスクの１モジュール、単一のタスクまたは単一のスレッドの実行を停止させ、前にこれらのために確立されたデータ通信モジュール、タスク間制御ブロックおよび直接メモリ・アクセス接続から切断するコマンドである。

【０１１０】図３５からわかるように、DSPMGR APIコントローラ４６１は、カテゴリー２のコマンドをタスク・マネージャ４５９に通信する。これを、図４１にブロック図形式で詳細に示す。ここで図４１を参照すると、制御ブロック４８７が、DSPMGRAPIコントローラ４６１からAPIコマンドを受け取り、これらを、ディジタル信号プロセッサ内のタスクの管理に関連する特定のタスクを実行するのに適切なモジュールおよびファイルに経路指定する。図からわかるように、モジュール、タスクまたはスレッドの解放を必要とするコマンドは、モジュール解放ファイル４８９、タスク解放ファイル４９１およびスレッド解放ファイル４９３のうちの適切なファイルに経路指定される。同様に、モジュールおよびタスクの状態変化を必要とするAPIコマンドは、モジュール状態変更ファイル５０５およびタスク状態変更ファイル５０７のうちの適切なファイルに経路指定される。サイクル毎フレームの変更を必要とするコマンドは、サイクル毎フレーム変更ファイル５０１に渡される。データ通信モジュール、直接メモリ・アクセスまたはタスク間制御ブロックの接続または切断を必要とするAPIコマンドは、DCM制御ファイル４９７、DMA制御ファイル５０３およびITC

B制御ファイル５１１のうちの適切なファイルに経路指定される。サイクル毎フレームの変更を要求するAPIコマンドは、フレーム・マネージャ制御ファイル４９５に経路指定される。DCM制御ファイル４９７とフレーム・マネージャ制御ファイル４９５は、ディジタル信号プロセッサBIOSを介して渡される前に、ディジタル信号プロセッサ・オペレーティング・システム制御ファイル４９９を通るように経路指定されることに留意されたい。

【０１１１】図４７からわかるように、APIコマンドのカテゴリー３には、３つのコマンドが含まれる。dsp Memtransfer（DSPメモリ転送）コマンドは、データと命令をデータ・メモリ４３と命令メモリ４１（共に図２）に書き込むことができるか、そこから読み取ることができるかを判定するコマンドである。dsp Reset（DSPリセット）コマンドは、ディジタル信号プロセッサを停止させ、ロケーション・カウンタを０にリセットするコマンドである。dsp Run（DSP走行）コマンドは、プロセッサの走行を開始するコマンドである。図３５からわかるように、これらのコマンドは、DSPMGR APIコントローラ４６１とDSP BIOSインターフェース４６５の間で通信される。その詳細を、ブロック図形式で図３９に示す。

【０１１２】ここで図３９を参照すると、DSP BIOSインターフェース４６５には、互いに協力してAPIコマンドが要求したタスクを実行する複数のブロックまたはファイルが含まれる。すべてのAPIコマンドが、制御ブロック５１３によって受け取られる。メモリ転送に関連するAPIコマンドは、メモリ転送ブロック５１５に向けられる。命令の読取を必要とする動作は、命令読取ファイル５１７に経路指定される。命令の書込を必要とする動作は、命令書込ファイル５１９に経路指定される。データの読取を必要とする動作は、データ読取ファイル５２１に向けられる。データの書込を必要とする動作は、データ書込ファイル５２３に向けられる。読取機能と書込機能のすべてが、入出力読書制御ブロック５３５を介して実行される。プロセッサのリセットを要求するAPI命令は、プロセッサ・リセット・ファイル５２５に向けられる。プロセスの走行開始を要求するAPI命令は、プロセッサ走行ファイル５２７に経路指定される。図３９からわかるように、カテゴリー５タイプのコマンドのうち、ディジタル信号プロセッサの能力に関して問い合わせるコマンドは、ディジタル信号プロセッサの構成と能力を判定するためにDSPBIOSインターフェース４６５を介して照会能力５２９に経路指定される可能性がある。図３９では、DSP BIOSインターフェース４６５が、割込みハンドラ・ファイル５３３を介して割込みを受け取ることができ、この割込みハンドラ・ファイル５３３は、割込みハンドラ導入ファイル５３１によって確立される。割込みハンドラ・ファイル５３３は、IPCハンドラ４６３に経路指定されるコールバックを生成する。IPCハンドラ４６３は、図３５に示され、図３７にブロック図形式で詳細に示されている。

【０１１３】ここで図３７を参照すると、IPCハンドラ４６３が、協力してIPCハンドラ４６３のタスクを実行する機能ブロックおよびファイルとして図示されている。割込み元タスク識別ブロック５３７が、実際に割込みを受け取り、割込みレジスタ５４９をアクセスして、特定の割込みに対して識別される特定のユーザ・コールバックを判定する。割込みレジスタ５４９に示されているように、割込みレジスタには、各ビットが要求された特定のコールバックを表す１６ビット・ワードが含まれる。１つまたは複数の割込みを同時に活動状態にすることができるが、割込みのそれぞれについて１つのユーザ・コールバックの生成が必要である。DSPマネージャは、１時に１ビットずつ１６ビット・ワード内をサイクルし、割込みとそれに対応するユーザ・コールバックのそれぞれに個別に対処する。コールバック検証ファイル５３９は、割込み元タスク識別ブロック５３７とコールバック・ディスパッチ・ファイル５４１の両方と通信する。コールバック・ディスパッチ・ファイル５４１は、実際にユーザ・コールバック信号を生成する。IPC検証ファイル５４７は、dsp connect IPC（DSP IPC接続）とdsp disconnect IPC（DSPIPC切断）を含む２種類のAPIコマンドを受け取る。dsp connect IPCコマンドによって、IPCハンドラ４６３が接続され、dsp disconnect IPCコマンドによって、IPCハンドラ４６３が切断される。IPC検証ファイル５４７は、コールバック・セット・ファイル５４５およびコールバック・クリア・ファイル５４３と通信する。コールバック・セット・ファイル５４５は、特定の割込みのために特定のコールバックを確立するのに使用される。コールバック・クリア・ファイル５４３は、前の割込みとコールバックの配置を無効化し、割込みとコールバックのプロトコルを再構成できるようにする。

【０１１４】図３６は、DSPMGR APIコントローラ４６１の詳細なブロック図である。図からわかるように、API制御ブロック５７９は、ディジタル信号プロセッサ・マネージャからAPIコマンドを受け取り、これを適切なタスク・ブロックに経路指定する。カテゴリー５のコマンドは、API制御ブロック５７９によって情報照会ブロック５８１に向けられる。ここで図４７を参照すると、カテゴリー５のコマンドには、「ハウスキーピング」コマンドとみなすことのできる複数のコマンドが含まれる。dsp abilities（DSP能力）コマンドは、そのコマンドが行われた時点でどの資源を使用可能であるかを判定するコマンドである。dsp label to addressコマンドは、メモリ転送をセットアップできるようにするためにドライバによってDSPマネージャに対して行われる呼出しである。dsp name to module handleコマンド、dsp name to task handleコマンドおよびdsp name to thread handleコマンドは、図３のTAMドライバ６３、サウンド・ドラ

イバ65、モデム・ドライバ67および他のディジタル信号処理ドライバ69と図3のDSPマネージャ71の間でのモジュール・レベル、タスク・レベルまたはスレッド・レベルでの通信を容易にするコマンドであり、通常のコマンドである。dsp query（DSP照会）コマンド、dsp info（DSP情報）コマンド、dsp query manager info（DSPマネージャ情報照会）コマンド、dsp query module info（DSPモジュール情報照会）コマンドおよびdsp miscellaneous info（DSP雑情報）コマンドは、ディジタル信号プロセッサ、DSPマネージャおよび選択されたモジュールに関係する情報の転送に使用されるコマンドである。これらの照会は、トラブルシューティングとハウスキーピング動作に特に有用である。

【0115】図35のDSPマネージャ71を用いると、ディジタル信号プロセッサ資源の動的でリアルタイムな連続的管理が可能になり、その結果、モジュラー・マルチメディア・ソフトウェア・タスクの動的または一時的なオーバーラップするロードと解放が可能になる。従来技術では、他のマルチメディア・ソフトウェア・タスクが実行されている間に特定のマルチメディア・ソフトウェア・タスクをロードまたは解放することが不可能であった。これが本発明によって実現される。

【0116】図42からわかるように、DSP資源マネージャ455には、ディジタル信号処理資源を独立に管理する複数のソフトウェア・ファイルが含まれる。ハードウェア使用可能度管理ファイル565は、マルチメディア末端装置と他の周辺装置の使用可能度を管理する。命令記憶域管理ファイル567は、図2の命令メモリ41での命令の記憶を独立に管理する。データ記憶域管理ファイル569は、図2のデータ・メモリ43でのデータの記憶を独立に管理する。命令記憶域管理ファイル567とデータ記憶域管理ファイル569から、常に、マルチメディア・ソフトウェア・タスクによって要求された命令メモリ41とデータ・メモリ43内のメモリの量と、追加のモジュラー・マルチメディア・ソフトウェア・タスクがディジタル信号プロセッサにロードされる時にデータと命令の受け取りのために使用可能な命令メモリとデータ・メモリの量とがわかる。サイクル毎秒管理ファイル571は、ディジタル信号プロセッサの消費されたサイクル毎秒と使用可能サイクル毎秒を継続的に監視する。このファイルから、常に、ディジタル信号プロセッサによって実行されているタスクが消費する毎秒サイクル数ならびに使用可能な毎秒サイクル数がわかる。十分な毎秒サイクルが使用可能でない限り、モジュラー・マルチメディア・ソフトウェア・タスクを実行のためディジタル信号プロセッサにロードすることはできない。

【0117】次に、直接メモリ・アクセス管理ファイル575は、直接メモリ・アクセス動作専用のディジタル信号プロセッサ資源を管理する。具体的に言うと、直接メモリ・アクセス管理ファイル575によって、直接メモリ・アクセス動作専用にされた命令メモリ41とデータ・メモリ43のメモリの量が判定される。十分な資源が使用可能でない限り、他の直接メモリ・アクセス・タスクは許可されない。IPC管理ファイル577は、使用可能な割込みの割振りを継続的に監視する。使用可能でない割込みを必要とするモジュラー・マルチメディア・タスクは、ディジタル信号プロセッサにロードされない。最後に、バス帯域幅管理ファイル573は、主データ通信バスと命令通信バスの活動のレベルを判定し、特定のタスクがバスの過度の使用を要求するかどうかを判定する。

【0118】ディジタル信号処理管理プログラムによって実現される重要な長所は、（1）このシステムを用いると、モジュラー・マルチメディア・ソフトウェア・タスクを、ディジタル信号プロセッサによって実行中の他のモジュラー・マルチメディア・タスクに割り込まずに開始でき、終了できることと、（2）そのモジュラー・マルチメディア・タスクをディジタル信号プロセッサによる並行実行に特に適合させる必要がないことである。この結果は、DSPマネージャ71を使用して、ディジタル信号プロセッサにロードされる可能性のあるタスクの受入れと実行に関してディジタル信号プロセッサの能力を動的に判定することによって達成される。この処理は、図35と図48を同時に参照することによって最もよく理解できる。この処理は、ブロック591から始まる。ブロック593で、マルチメディア・アプリケーションによってDSPマネージャ71にAPIロード・コマンドが供給される。このコマンドは、複数のモジュラー・マルチメディア・ソフトウェア・タスク、単一のモジュラー・マルチメディア・ソフトウェア・タスクまたは単一の実行スレッドからなるモジュールを、ディジタル信号プロセッサによる実行のためにディジタル信号プロセッサへロードするよう要求するコマンドである。ブロック595で、ディジタル信号プロセッサへのロードを要求されたタスクまたはモジュールを、DSPパーサ451が受け取り、図38に示されるように解析する。図38によれば、ファイルは、ファイル読取バッファに読み込まれ（ブロック583）、ブロック585で、さまざまな通常の「チャンク」または「テーブル」に解析される。図38のブロック587で、DSPパーサ451が、解析された要素のために必要なデータ構造を作成する。

【0119】図48に戻って、ブロック597で、ディジタル信号プロセッサへのロードを要求された解析済みのタスクまたはモジュールを、タスク・ローダ457によって検査して、その「要件」を判定する。要件は、タスクまたはモジュールが要求する命令メモリおよびデータ・メモリの量、タスクまたはモジュールによって実行時に消費される毎秒サイクル数、タスクまたはモジュールが（データ通信モジュールを介して）読み書きする接

続の個数、および、タスクまたはモジュールの実行中に必要になる割込みがあれば、その割込みの個数である。

【0120】ブロック599で、図42に示されたDSP資源マネージャ455に相談して、タスクまたはモジュールのロードが可能であるかどうかを判定する。図42に示されるように、DSP資源マネージャ455には、ハードウェア使用可能度、命令メモリ使用可能度、データ・メモリ使用可能度、サイクル毎秒使用可能度、バス帯域幅使用可能度、割込みベクトル使用可能度および直接メモリ・アクセス資源使用可能度を独立に監視する別々のファイルが含まれる。ハードウェア使用可能度管理ファイル565は、マルチメディア末端装置の使用可能度を継続的に監視し、これらのマルチメディア末端装置が、ディジタル信号プロセッサが実行中の他のモジュラー・マルチメディア・ソフトウェア・タスクによって使用されているかどうかの表示を提供する。命令記憶域管理ファイル567は、ディジタル信号プロセッサが実行中のモジュラー・マルチメディア・ソフトウェア・タスクによって使用されている命令メモリの量を継続的に監視し、また、ディジタル信号プロセッサへのロードのために要求される可能性のある、モジュラー・マルチメディア・ソフトウェア・タスクのために使用可能な命令メモリの総量を継続的に判定する。

【0121】次に、データ記憶域管理ファイル569は、データ・メモリを継続的に監視して、ディジタル信号プロセッサが実行中のモジュラー・マルチメディア・ソフトウェア・タスクによって現在使用されているデータ・メモリの総量を判定する。また、データ記憶域管理ファイル569は、実行のためディジタル信号プロセッサへのロードのために要求される可能性のある、ファイルに使用可能なデータ・メモリの総量を継続的に監視する。サイクル毎秒管理ファイル571は、ディジタル信号プロセッサが実行中のモジュラー・マルチメディア・タスクによって消費されつつあるディジタル信号プロセッサのサイクル毎秒を継続的に監視する。サイクル毎秒管理ファイル571は、実行のためディジタル信号プロセッサにロードされる可能性がある他のモジュラー・マルチメディア・ソフトウェア・タスクのために使用することのできる総使用可能サイクル毎秒の表示を継続的に提供する。バス帯域幅管理ファイル573は、ディジタル信号プロセッサ内のデータ・バスの使用可能度を継続的に監視し、ディジタル信号プロセッサのバス容量の現在表示を提供して、追加のモジュラー・マルチメディア・タスクを実行のためディジタル信号プロセッサにロードすることができるかどうかの判定を可能にする。IPC管理ファイル577は、割込みベクトルの割振りと使用可能度を継続的に監視する。

【0122】図27に示されるように、好ましいディジタル信号プロセッサには、専用の割込みレジスタが含まれ、この割込みレジスタを用いて、ディジタル信号プロセッサが実行中のさまざまなモジュラー・マルチメディア・タスクによって16個の異なる割込みベクトルを使用できる。ディジタル信号プロセッサが同時に16個のモジュラー・マルチメディア・ソフトウェア・タスクを実行しており、各タスクが1つの割込みベクトルを必要とする場合、割込みベクトルの使用を必要とする追加のモジュラー・マルチメディア・ソフトウェア・タスクをディジタル信号プロセッサにロードすることはできない。もちろん、割込みベクトルの使用を必要としないタスクであれば、ディジタル信号プロセッサに追加のモジュラー・マルチメディア・ソフトウェア・タスクをロードして実行できる。

【0123】図48と図35に戻ると、DSP資源マネージャ455に相談して、ハードウェア使用可能度、命令メモリ使用可能度、データメモリ使用可能度、サイクル毎秒使用可能度、バス帯域幅使用可能度、直接メモリ・アクセス使用可能度および割込みベクトル使用可能度に関する現在の資源割振りと使用可能度を考慮して、タスクまたはモジュールのロードが可能であるかどうかを判定する。特定のモジュラー・マルチメディア・ソフトウェア・タスクが、ディジタル信号プロセッサへのロードのためにマルチメディア・アプリケーション・ソフトウェアによって呼び出される場合、そのタスクは、使用可能資源またはディジタル信号プロセッサを超えてはならない。たとえば、特定のモジュラー・マルチメディア・ソフトウェア・タスクによって操作されるマルチメディア末端装置が使用可能であり、十分な命令メモリとデータ・メモリの空間が使用可能であるが、十分なディジタル信号プロセッサ・サイクル毎秒が使用可能でない場合、そのタスクは、ディジタル信号プロセッサにロードされない。

【0124】別の例として、特定のモジュラー・マルチメディア・ソフトウェア・タスクが、ディジタル信号プロセッサへのロードを要求されたが、そのディジタル信号プロセッサが実行中の他のモジュラー・マルチメディア・ソフトウェア・タスクによって16個の割込みベクトルのすべてが現在使用されている場合、このロード動作は許可されない。ロードが許可されると判定される場合、そのタスクまたはモジュールは、ブロック601でDSPリンク・エディタ453に送られて、モジュール・フィックスアップ・ブロック553（図40）、タスク・フィックスアップ・ブロック555およびセグメント（またはスレッド）フィックスアップ・ブロック557での「フィックスアップ」動作を行って、ブロック559およびブロック561によるデータ・メモリおよび命令メモリへの書込を可能にする。ブロック603で、タスクまたはモジュールをDSPの命令メモリとデータ・メモリにロードする。

【0125】この形で、単一のディジタル信号プロセッサに、ディジタル信号プロセッサ資源の使用可能度だけ

を考慮して、さまざまなモジュラー・マルチメディア・ソフトウェア・タスクをロードすることができる。この特徴の結果として、モジュラー・マルチメディア・ソフトウェア・タスクは、他のモジュラー・マルチメディア・ソフトウェア・タスクの存在または実行を考慮に入れるように構成、修正または配置する必要がない。これによって、単一のディジタル信号プロセッサによって同時にロードし実行することのできる完全に独立なマルチメディア・アプリケーションをエンド・ユーザが作成できるオープン・アーキテクチャがもたらされる。このオープン・アーキテクチャと設計のモジュール性によって、複数のディジタル信号プロセッサを「ネットワーク化」して、複数のモジュラー・マルチメディア・タスクの実行を共用することも可能になる。図４９に示されるように、N個のディジタル信号プロセッサを、単一のDSPマネージャと通信状態にすることができる。図からわかるように、ディジタル信号プロセッサ６１１、ディジタル信号プロセッサ６１３、ディジタル信号プロセッサ６１５およびディジタル信号プロセッサ６１７は、並列に動作して、複数のモジュラー・マルチメディア・ソフトウェア・タスクを実行することができる。

【０１２６】複数のディジタル信号プロセッサを、DSPマネージャ７１などの単一のDSPマネージャと共に使用する時、DSPマネージャは、ディジタル信号プロセッサのうちの特定の１つにタスクをロードしようと試みる。あるディジタル信号プロセッサが、特定のモジュールまたはタスクのロードに使用できない場合、DSPマネージャ７１は、次のディジタル信号プロセッサへのロードを試みる。このディジタル信号プロセッサが十分な資源を有しない場合、DSPマネージャは、さらに別のディジタル信号プロセッサへのロードを試みる。この処理は、DSPマネージャ７１によってロードを要求されたタスクまたはモジュールを受け入れるのに十分な資源を有する使用可能なディジタル信号プロセッサが見つかるまで繰り返される。これによって、既存のシステムに多数のディジタル信号プロセッサを追加して、マルチメディア・アプリケーションの処理能力を高めることができる、設計のモジュール性ももたらされる。

【０１２７】ここで図５０を参照すると、タスクをロードする処理が示されており、このシステムは、ブロック７０１でパワー・アップされる。その後、ブロック７０３で、DSPの個数を判定する。この処理では、まずそのシステムで何個のDSPが使用可能であるかを判定し、したがって、どれだけの資源を使用しなければならないかを判定できるようにする。各DSPは、バス上にそれ自体の入出力アドレス範囲を有するので、各アドレスを照会してDSPが存在するかどうかを調べるのは簡単である。ブロック７０５で、DSPを初期設定する。この処理は、周知であり、単に、コマンドおよびタスクのロードの前置きとして全レジスタ値のリセットと全メモリのクリアを行うだけである。次に、ブロック７０７で、メモリからロードする機能を識別する。アプリケーション・プログラムは、そのAPI（アプリケーション・プログラム・インターフェース）を介して、DSPに機能をロードするようDSPマネージャに伝える。「機能」は、複数の「タスク」から構成される。その後、ブロック７０９で、システム・メモリからロードするタスクを識別する。機能は複数のタスクから構成されるので、アプリケーションは、DSPにロードする必要がある所与の機能の最初のタスクを識別する。ブロック７１１で、タスク平衡化を実行して、タスクの行き先を決定する。タスク間のリンケージ要件（データ通信モジュール（DCM)の個数とサイズ）およびタスクを分散できるかどうか（タイミング要件に基づく）と、さまざまな入出力インターフェースの使用可能度および接続位置と、残っているDSP命令メモリ量およびデータ・メモリ量と、各DSP上でまだ使用可能なMIPS（百万命令毎秒）の量と、通信方法と、システム上で使用可能な資源とを、本発明の好ましい実施例に従って考慮に入れることができる。"MIPS"は、処理能力の尺度である。

【０１２８】DSP間通信は、選択された複数DSPアーキテクチャに応じて複数の方法によって達成できる。DSPが互いに通信するための方法として、直列ポート、局所化された相互接続バスまたは共用メモリのどれが使用されるかに基づいて、異なる資源値が見つかる。これらの異なる方法のそれぞれが、それ自体の伝送特性と動作方式を有するので、全体的なアルゴリズムは、どのシステムを使用するかに基づいて変化する可能性がある。本発明の独自の態様は、システム特性またはサブシステム特性がさまざまな負荷に伴って変化するにつれて、タスクの配分を変更でき、必ずタスクの最適配置がもたらされることである。タスク平衡化は、下で図５１に示された流れ図を参照して詳細に説明する。

【０１２９】次に、ブロック７１３で、ホストのメモリ・マップとタスク・マップを編成する。このブロックでは、DSPマネージャが各タスクの記憶位置と使用位置の情報を示すことによって、メモリ・マップを構築する。各タスクは、アプリケーション作者によって、本発明の好ましい実施例に従って、MIPS、メモリ必要量、使用する割込みおよびチャネルの既知の値を用いて事前定義されている。ブロック７１５で、タスクを適切なDSPにダウンロードする。下記の図５１のタスク平衡化処理がこのタスクをどこに置くよう命ずるかに基づいて、その作業を実行するDSPの命令メモリおよびデータ・メモリにタスクをダウンロードする。その後、ブロック７１７で、DCMとITCBのためのDSPデータ・メモリを割り振る。次に、ブロック７１９で、特定のデーモンを適切なDSPにダウンロードする。デーモンは、通信チャネルを介するDSP間通信の実現に使用される。デーモンは、DCMのポインタも更新するが、これについては上で説明した。こ

のブロック７１９では、他のデータがDCM用の空間を上書きしないように、DCM用の空間を記憶する。この時点で、命令プログラムがロードされており、相互通信デーモンが特性設定され、位置をロードされており、メモリがマップに割り振られている。この時点で、システムは動作の準備が調っている。その後、ブロック７２１で、別のタスクをロードするかどうかの判定を行う。すべてのタスクをロードするまで、ブロック７０９へのループ・バックを継続する。すべてのタスクをロードした後には、ブロック７２３で機能を走行させる。機能は複数のタスクから構成されるので、機能を走行させる前に他のタスクをロードする必要があるかどうかを調べることが適切である。ブロック７２５で、追加の機能をロードしなければならないかどうかの判定を行う。追加の機能をロードする必要がなくなるまで、ブロック７０７に戻る。ブロック７２５で、ユーザ入力を検査して、ユーザが動作前または動作中に別の機能の追加を決定したかどうかを判定する。追加の機能をロードする必要がなくなった時には、ブロック７２７で、追加のコマンドを待つ。

【０１３０】ここで図５１を参照すると、「局所化された相互接続バス」とも称する「ローカル・バス」によって互いに接続されたDSPのタスク平衡化のための処理の流れ図が示されている。ブロック７３１で、タスク平衡化の処理が始まる。DSPマネージャは、アプリケーション・プログラムによって要求される機能とそれを構成するタスクを識別し終えている。この時点で、MIPS、命令メモリ必要量、データ・メモリ必要量、DCMのタイプおよびサイズなどのタスクの特性を考慮する。ブロック７３３で、DCMが他のタスクに接続されるかどうかの判定を行う。タスクを他のDSPに割り当てられるようになる前に、DSPが、所与のDSP上に既に配置されているロードすべきタスクと同一の機能を構成するタスクにDCMを接続したかどうかを判定することが重要である。他のタスクと共用しなければならないデータを有しない場合、そのタスクはどのDSPに割り当ててもかまわない。他のタスクに接続されたDCMがある場合、タスクを特定のDSPに置く前に、より多くの情報を判定しなければならない。

【０１３１】DCMが他のタスクに接続される場合、ブロック７３５で、これらのDCMがリアルタイムDCMであるかどうかの判定を行う。リアルタイム接続は、タスクが互いにすばやく効率的に通信しなければならず、わずかな通信遅延だけを有する必要があることを意味する。確立しようとしているDCMがリアルタイムでない場合、通信チャネル（すなわち、DSPを相互接続するローカル・バス）が、チャネル上の他の活動に割り込まずにこの追加の通信を成功裡に処理できるかどうかに関する検査を行う。この判定は、どのタスクがどのDSPに既にロードされているかと、ローカル・バス上でどれほどの通信トラフィックが発生するかと、そのチャネル自体の本来の能力とを知っているDSPマネージャ内で走行する小さいサブルーチンで行われる。DCMがリアルタイムDCMである場合、ブロック７３７で、DSPに置かれたDCMに基づいて、DSPを選択する。この処理では、ロードされるタスクと同一の機能の一部であるタスクに接続されたDCMを含むDSPを選択する。

【０１３２】次に、ブロック７３９で、このDCM接続を有するDSP上に十分なMIPSがあるかどうかの判定を行う。MIPS単位で測定されるプロセッサ・パワーは、複数の異なる方法で判定できる。Mwave DSPは、International Business Machines Corporation社とTexas Instruments, Inc.社から入手可能であり、使用可能MIPSに関するデータを取得することができるレジスタを有する。その代わりに、各DSP上で最低優先順位のプログラムまたはタスクとしてデーモンを走行させることができる。このデーモンが走行した時間の量を使用して、使用可能なMIPSまたは処理能力を示すことができる。これらのデーモンからのデータは、DSPマネージャが要求を送り、その結果、このデータ要求を、DSPによる処理のためにスケジューリングさせることによって要求できる。非常に大量のタスクまたはタスクの集合が、既に１つのDSPに置かれている可能性があるので、通信の観点からこれを行うとしても、別のタスクをロードするために使用可能な処理能力や処理サイクルが不十分な場合がある。したがって、このステップは、その可能性を検査するために必要である。この特定のDCM接続を有するDSP上で十分なMIPSが使用可能な場合、そのDSP上に十分な命令メモリとデータ・メモリが存在するかどうかの判定を、ブロック７４１で行う。そのDSP上に十分な命令メモリとデータ・メモリが存在する場合、ブロック７４２で、このDSPをこのタスク用のDSPとして識別する。その後、ブロック７１４で処理を終了する。

【０１３３】ブロック７３９およびブロック７４１に戻って、十分なMIPSまたは十分な命令メモリおよびデータ・メモリが存在しない場合、ブロック７４３で、追加のデータ・フローを扱うのに十分なチャネル資源が使用可能であるかどうかの判定を行う。この時点で、元々選択されたDSPにタスクをロードできないので、DSP間チャネルが新しいDCMの通信要件をサポートできるかどうかを調べるための検査を行う。使用可能なチャネル資源が不十分な場合、ブロック７４５で、タスクをロードできないことを伝えるメッセージをオペレーティング・システムに送る。ブロック７４３に戻って、十分なチャネル資源を使用できる場合、ブロック７４７で、すべてのDSPのMIPSを比較し、最大量のMIPSを使用可能なDSPを選択する。このブロックを実行するのは、接続を必要とするDCMが存在しないか、DSP間通信を処理するのに十分なチャネル資源が存在するからである。この時点で、すべてのDSPを比較して、最大の使用可能MIPSが残っているDSPを調べ、最大の使用可能MIPSが残っているDSPを選択す

る。ブロック７４９で、このDSPにタスクをロードするのに十分なMIPSが使用可能であるかどうかの判定を行う。十分なMIPSが存在する場合、ブロック７５１で、このDSPに十分な命令メモリとデータ・メモリが存在するかどうかの判定を行う。命令メモリとデータ・メモリが十分な場合、ブロック７５４で、このDSPをそのタスク用のDSPとして識別し、ブロック７５６で処理を終了する。ブロック７５１に戻って、十分な量のMIPSが使用可能でないか、DSP上に十分な命令メモリおよびデータ・メモリが存在しない場合には、ブロック７５３で、他のDSPが存在するかどうかを判定する。ブロック７４９に戻って、十分な量のMIPSが使用可能でない場合、ブロック７４５に進んで、タスクをロードできないことを伝えるメッセージを送る。

【０１３４】他のDSPが存在しない場合、ブロック７４５で、タスクをロードできないことを伝えるメッセージをオペレーティング・システム送る。他のDSPが存在すると、ブロック７５７で、次に大きい使用可能MIPSを有するDSPが選択されることになる。その後、処理はブロック７４９に戻る。ブロック７３５に戻って、リアルタイムDCMが存在しない場合、ブロック７５９で、このタスクを別のDSPに置いた場合に十分なチャネル帯域幅が存在するかどうかを判定する。接続するDCMがリアルタイムではなく、DSPと他のすべてのトラフィックの間の通信を扱うのに十分なチャネル帯域幅がある場合、どのDSPがこのタスクの走行に最適であるかを判定する部分のコードに移る。十分なチャネル帯域幅がない場合、検査を行って、このDCMが接続されるタスクを既に有するDSPにこのタスクをロードできるかどうかを調べる。十分なチャネル帯域幅が存在する場合、上で述べたように処理はブロック７３９に進む。十分なチャネル帯域幅が存在すると、前に述べたように処理がブロック７４７に進むことになる。ブロック７３３に戻って、DCMが他のタスクに接続されない場合、処理は上で述べたブロック７４７に進む。

【０１３５】ここで図５２を参照すると、前にロードされ走行中の機能を除去するための処理の流れ図が示されている。ブロック７７１で、タスクをリストする。タスクを除去するには、どのタスクが動作中であるか、どのメモリがそれらのタスクに割り振られているか、どのDCMが存在するか、どのDSPにタスクが置かれているかをリストしなければならない。言い換えると、除去しようとするタスクの明細を知る必要がある。ブロック７７３で、このリストからの１タスクが、DSPオペレーティング・システム内のフレームから除去され、このタスクは、フレーム・マネージャ（元のDSPマネージャのサブセクション）にこのタスクを無視するように命令することによって、フレームから除去されなければならない。このタスクは、今後はこのフレームの活動が走行する時に処理されなくなる。この処置は、単一DSP環境でも複数DSP環境でも同一になるはずである。その後、ブロック７７５で、タスクを停止させ、その後、ブロック７７７で、DCMとITCBに関するメモリ・マークを除去する。影響を受けるDCMに関連するメモリの開始点、終了点および中間位置を、データ・メモリから除去し、これによってこれらの位置を他の活動のために解放しなければならない。次に、ブロック７７９で、ホストのメモリ・ツリーからタスクを除去する。データ・メモリ空間を解放した後に、このタスクに関連する命令空間も割振り解除する。この特徴を用いると、DSPマネージャが、DSPのそれぞれの作業負荷を監視し、動的に維持できるようになる。ブロック７８１で、リスト上に他のタスクがあるかどうかの判定を行う。通常は１機能について複数のタスクが存在するので、１時に除去すべきタスクが複数存在する。他の機能は、タスクが除去されている間であっても継続する。すべてのタスクが除去されるまで、ブロック７７３に戻ってこの処理を継続する。その後、ブロック７８３で処理を停止する。タスクをロードする時には、図５０のブロック７０７ないしブロック７２７に記載の処理に従う。

【０１３６】まとめとして、本発明の構成に関して以下の事項を開示する。

【０１３７】（１）各機能が少なくとも１つのタスクから構成される、データ処理システム内の複数のディジタル信号プロセッサによる機能の実行を管理するためのデータ処理システムであって、データ処理動作用の中央処理装置と、前記中央処理装置に接続され、通信チャネルによって互いに接続された、複数のディジタル信号プロセッサと、ディジタル信号プロセッサ・マネージャとを含み、前記ディジタル信号プロセッサ・マネージャが、ディジタル信号プロセッサのために存在するプロセッサ資源を識別するための第１識別手段と、機能がディジタル信号プロセッサにロードされることの表示に応答して、前記機能の一部を構成する、ディジタル信号プロセッサにロードされるタスクを識別するための第２識別手段と、機能が除去されることの表示に応答して、前記機能の一部を構成する、ディジタル信号プロセッサ上の、ディジタル信号プロセッサから除去されるタスクを識別するための第３識別手段と、第１識別手段と、追加されるタスクを識別する第２識別手段とに応答して、識別されたタスクの実行をサポートするのに十分な資源を有するディジタル信号プロセッサを選択するための選択手段と、選択手段によるディジタル信号プロセッサの選択に応答して、ディジタル信号プロセッサ上で実行中の別の機能を構成するタスクのいずれの実行にも割り込まずに、追加すべきタスクを選択されたディジタル信号プロセッサにロードするためのロード手段と、除去すべきタスクを識別する第３識別手段に応答して、ディジタル信号プロセッサ上で実行中の別の機能を構成するタスクのいずれの実行にも割り込まずに、識別されたタスクをデ

ィジタル信号プロセッサから除去するための除去手段とを含む前記ディジタル信号プロセッサ・マネージャとを含むことを特徴とするデータ処理システム。

（２）前記除去手段が、除去すべき識別されたタスクをフレームから除去する手段と、除去すべき識別されたタスクの実行を停止させる手段と、除去すべき識別されたタスクに関連するデータ通信モジュールのそれぞれのメモリ・マークを除去する手段と、除去すべき識別されたタスクをメモリ・ツリーから除去する手段とを含むことを特徴とする、上記（１）のデータ処理システム。

（３）データ処理システム内の複数のディジタル信号プロセッサによるタスクの実行を管理するための前記データ処理システムであって、データ処理動作用の中央処理装置と、通信チャネルによって互いに接続され、前記中央処理装置に接続された、複数のディジタル信号プロセッサと、ディジタル信号プロセッサ・マネージャとを含み、前記ディジタル信号プロセッサ・マネージャは、ディジタル信号プロセッサによって実行される機能の識別に応答して、機能の一部を構成するタスクをロードのために識別するための第１識別手段と、機能の一部を構成するロードされたタスクに接続されたデータ通信モジュールが存在するかどうかを識別するための第２識別手段と、接続されたデータ通信モジュールの存在の識別に応答して、接続されたデータ通信モジュールがリアルタイム・データ通信モジュールであるかどうかを判定する第１判定手段と、リアルタイム・データ通信モジュールの存在に応答して、識別されたタスクをサポートするのに十分なプロセッサ資源が、データ通信モジュールを含むディジタル信号プロセッサ用に存在するかどうかを判定するための第２判定手段と、リアルタイム通信モジュールを含む前記ディジタル信号プロセッサ用の十分なプロセッサ資源の存在に応答して、識別されたタスクをディジタル信号プロセッサに追加するための追加手段と、リアルタイム通信モジュールを含む前記ディジタル信号プロセッサ用の十分なプロセッサ資源の不在またはリアルタイム・データ通信モジュールの不在のいずれかに応答して、通信チャネルが追加のデータ通信モジュールをサポートするのに十分な通信資源を有するかどうかを判定するための第３判定手段と、追加のデータ通信モジュールをサポートするのに十分な通信資源の存在またはデータ通信モジュールの不在のいずれかに応答して、識別されたタスクをサポートする能力を有する、最大量のプロセッサ資源を有するディジタル信号プロセッサを選択するための手段と、ディジタル信号プロセッサの選択に応答して、選択されたディジタル信号プロセッサに識別されたタスクをロードするための手段とを含む前記ディジタル信号プロセッサ・マネージャを含み、１機能を構成するタスクを、ディジタル信号プロセッサ上で実行中の別の機能を構成するタスクの実行に割り込まずに、ディジタル信号プロセッサにロードできることを特徴とするデータ処理システム。

（４）ディジタル信号プロセッサからの除去のため、ディジタル信号プロセッサ上で実行中の機能の一部を構成するタスクを識別するための第４識別手段と、別の機能を構成するタスクの実行に割り込まずに、ディジタル信号プロセッサから識別されたタスクを除去するための除去手段とをさらに含む、上記（３）のデータ処理システム。

（５）前記除去手段が、識別されたタスクを含むフレームから、識別されたタスクを除去するための手段と、識別されたタスクの実行を終了させるための終了手段と、データ通信モジュールへのメモリ・マークを除去するための手段と、ホストのメモリ・ツリーから識別されたタスクを除去するための手段とを含むことを特徴とする、上記（４）のデータ処理システム。

（６）各機能が少なくとも１つのタスクからなり、データ処理システムが、複数のディジタル信号プロセッサに接続された、データ処理動作用の中央処理装置を含み、ディジタル信号プロセッサが、通信チャネルによって互いに接続される、データ処理システム内の複数のディジタル信号プロセッサによる機能の実行を管理するための、データ処理システム内の方法であって、データ処理システムによって実施される、ディジタル信号プロセッサ用に存在するプロセッサ資源を識別するステップと、機能がディジタル信号プロセッサにロードされることの表示に応答して、その機能の一部を構成する、ディジタル信号プロセッサにロードされるタスクを識別するステップと、機能が除去されることの表示に応答して、その機能の一部を構成する、ディジタル信号プロセッサから除去されるディジタル信号プロセッサ上のタスクを識別するステップと、プロセッサ資源の識別と追加されるタスクの識別とに応答して、識別されたタスクの実行をサポートするのに十分な資源を有するディジタル信号プロセッサを選択するステップと、選択ステップによるディジタル信号プロセッサの選択に応答して、ディジタル信号プロセッサ上で実行中の別の機能を構成するタスクのいずれの実行にも割り込まずに、選択されたディジタル信号プロセッサに追加すべきタスクをロードするステップと、除去すべきタスクの識別に応答して、ディジタル信号プロセッサ上で実行中の別の機能を構成するタスクのいずれの実行にも割り込まずに、ディジタル信号プロセッサから識別されたタスクを除去するステップとを含む方法。

【０１３８】

【発明の効果】図５０および図５１に示された処理は、本発明の好ましい実施例によってDSPマネージャ７１に組み込まれる。本発明は、いずれも米国ニューヨーク州ArmonkのInternational Business Machines Corporation社の製品であるIBM PS/2コンピュータまたはIBM RISC SYSTEM/6000コンピュータなどの適当なホスト・プロセ

ッサを使用して実施できる。"RISC SYSTEM/6000"および"PS/2"はInternational Business Machines Corporation社の登録商標である。Mwaveにあるディジタル信号プロセッサなどのディジタル信号プロセッサを使用することができる。Mwaveは、単一ディジタル信号プロセッサを使用するマルチメディア・データ・ストリーム処理用の基板のファミリーである。Mwaveは、International Business Machines Corporation社の登録商標であり、International Business Machines Corporation社およびTexas Instruments, Inc.社によって供給される。Mwave用の現在入手可能なDSPマネージャは、米国マサチューセッツ州CambridgeのIntermetrics' Inc.社から入手可能なMwave Development Tool kit and the User's Guideを使用して、複数DSPを管理するように修正することができる。

【図面の簡単な説明】

【図１】複数のマルチメディア末端装置に接続され、マルチメディア・アプリケーションの処理に特に適したマルチメディア・データ処理システムを示す絵図である。

【図２】マルチメディア末端装置の動作を制御するマルチメディア・アプリケーションの実行に利用される主なハードウェア構成要素を示すブロック図である。

【図３】マルチメディア・ディジタル信号処理動作の主なソフトウェア構成要素を示すブロック図である。

【図４】複数のディジタル信号プロセッサの図を含む、図３の構成要素を示すもう１つのブロック図である。

【図５】ハードウェア・ディジタル信号プロセッサとそのそれぞれのインターフェース・ドライバおよびBIOSインターフェースの図を含む、図３の構成要素を示すもう１つのブロック図である。

【図６】DSP BIOSインターフェースと複数DSP資源マネージャと図５のインターフェース・ドライバの間でのインターフェースのステップを示す流れ図である。

【図７】図５のインターフェース・ドライバとDSP資源マネージャの間の割込み処理のステップを示す流れ図である。

【図８】DSP BIOSドライバの初期設定と複数のDSP BIOSドライバのロードのステップを示す流れ図である。

【図９】複数DSP資源マネージャの初期設定のステップを示す流れ図である。

【図１０】N個のモジュラー・マルチメディア・ソフトウェア・タスクのグループ化を表すモジュールからなるマルチメディア動作を示すブロック図である。

【図１１】マルチメディア末端装置と専用データ・メモリの間でのリアルタイム・データまたは非同期ストリーム化データの直接読み書きを示すブロック図である。

【図１２】モジュール化されたオープン・アーキテクチャを得るためのデータ通信モジュールとタスク間制御ブロックの使用を表すブロック図である。

【図１３】モジュール化されたオープン・アーキテクチャを得るためのデータ通信モジュールとタスク間制御ブロックの使用を表すブロック図である。

【図１４】モジュール化されたオープン・アーキテクチャを得るためのデータ通信モジュールとタスク間制御ブロックの使用を表すブロック図である。

【図１５】モジュラー・マルチメディア・ソフトウェア・タスクとマルチメディア末端装置の間でのデータ転送のためにデータ通信モジュールを秩序だった形で使用できるようにする好ましい動作規則を示す図である。

【図１６】モジュラー・マルチメディア・ソフトウェア・タスクとマルチメディア末端装置の間でのデータ転送のためにデータ通信モジュールを秩序だった形で使用できるようにする好ましい動作規則を示す図である。

【図１７】モジュラー・マルチメディア・ソフトウェア・タスクとマルチメディア末端装置の間でのデータ転送のためにデータ通信モジュールを秩序だった形で使用できるようにする好ましい動作規則を示す図である。

【図１８】本発明の好ましい実施例でデータ通信モジュールと共に使用するために確立された４つの標準化通信プロトコルの１つを表す絵図である。

【図１９】本発明の好ましい実施例でデータ通信モジュールと共に使用するために確立された４つの標準化通信プロトコルの１つを表す絵図である。

【図２０】本発明の好ましい実施例でデータ通信モジュールと共に使用するために確立された４つの標準化通信プロトコルの１つを表す絵図である。

【図２１】本発明の好ましい実施例でデータ通信モジュールと共に使用するために確立された４つの標準化通信プロトコルの１つを表す絵図である。

【図２２】最下位５ビットがデータ通信プロトコルの選択を表す、１６ビット・ワードを示す図である。

【図２３】オーナー・タスクとユーザ・タスクでの制御ブロックの作成を示す図である。

【図２４】複数のモジュラー・マルチメディア・ソフトウェア・タスクから複数のデータ通信モジュールを介して単一のユーザ・タスクに至る、仮想データ通信モジュールの使用を介するデータ・ストリームの通信を表すブロック図である。

【図２５】図２４の仮想データ通信モジュールの動作を示す、より詳細なブロック図である。

【図２６】複数のモジュラー・マルチメディア・ソフトウェア・タスクから単一のCODディジタル・アナログ変換器への、仮想データ通信モジュールの使用を介するオーディオ出力信号の多重化を示すブロック図である。

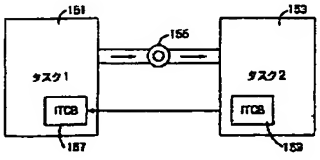
【図２７】タスク間制御ブロックの動作を示すブロック図である。

【図２８】同期式プロトコル・モードの動作でのデータ通信モジュールへの書込タスクを示す流れ図である。

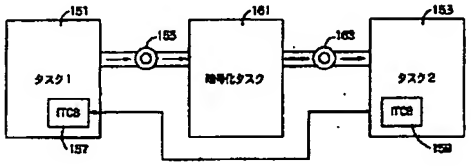
【図２９】同期式プロトコル・モードの動作でのデータ通信モジュールからの読取タスクを示す流れ図である。

【図３０】ユーザ・データ駆動プロトコル・モードの動作でのデータ通信モジュールへの書込タスクを示す流れ図である。
【図３１】オーナー・データ駆動プロトコルモードの動作または安全データ駆動プロトコル・モードの動作のいずれかでのデータ通信モジュールからのデータ読取タスクを示す流れ図である。
【図３２】そのうちの一部が通常のハードウェア・マルチメディア末端装置の動作を仮想化する、複数のモジュラー・マルチメディア・ソフトウェア・タスクの動作を示すブロック図である。
【図３３】オーディオ出力増幅タスクの動作を示すブロック図である。
【図３４】データ・ストリームの間の位相差を制御するためのデータ通信モジュールとモジュラー・マルチメディア・ソフトウェア・タスクの使用を示す、ブロック図である。
【図３５】図3のディジタル信号プロセッサ・マネージャ・プログラムを示すブロック図である。
【図３６】図３５のディジタル信号プロセッサ・マネージャAPI制御の詳細を示すブロック図である。
【図３７】図３５のIPCハンドラの詳細を示すブロック図である。
【図３８】図３５のDSPバーサーの詳細を示すブロック図である。
【図３９】図３５のDSP BIOSブロックの詳細を示すブロック図である。
【図４０】図３５のDSPリンク・エディタの詳細を示すブロック図である。
【図４１】図３５のタスク・マネージャの詳細を示すブロック図である。
【図４２】図３５の資源マネージャの詳細を示すブロック図である。
【図４３】図３５のDSPタスク・ローダーの詳細を示すブロック図である。
【図４４】データ通信モジュールの作成と定義に使用されるアセンブリ言語マクロ・テーブルを示す図である。
【図４５】データ通信モジュール・プロトコルの可能な組合せを示す表である。
【図４６】仮想データ通信モジュールの作成を可能にする制御ブロックに含まれる情報を表す表である。
＊【図４７】ホストCPUに常駐するディジタル信号プロセッサ・マネージャ・プログラムがディジタル信号プロセッサの動作を調整し制御できるようにする、アプリケーション・プログラム・インターフェース（API）の５つの大まかなカテゴリーを識別する表である。
【図４８】モジュラー・マルチメディア・タスクを実行のためにDSPにロードできるかどうかを判定する処理を示す流れ図である。
【図４９】モジュラー・マルチメディア・ソフトウェア・タスクを実行するための複数のDSPの「ネットワーク化」を示すブロック図である。
【図５０】タスクの初期設定と複数のDSPを含むデータ処理システムへの追加のための処理を示す流れ図である。
【図５１】タスク平衡化のための処理を示す流れ図である。
【図５２】前にロードされ走行中の機能を構成するタスクを除去するための処理を示す流れ図である。
【符号の説明】
４５１　ディジタル信号プロセッサ・バーサ
４５３　ディジタル信号プロセッサ・リンク・エディタ
４５５　ディジタル信号プロセッサ・資源マネージャ
４５７　タスク・ローダ
４５９　タスク・マネージャ
４６１　ディジタル信号プロセッサ・マネージャ・アプリケーション・プログラム・インターフェース・コントローラ
４６３　割込みおよびプロセッサ・コール・バック・ハンドラ
４６５　ディジタル信号プロセッサBIOSインターフェース
４６７　モジュール・ロード・ファイル
４６９　タスク・ロード・ファイル
４７１　スレッド・ロード・ファイル
４７３　制御ファイル
４７５　セグメント割振りファイル
４７７　マネージャ初期設定ファイル
４７９　データ・イメージ・ロード・ブロック
４８１　コード・イメージ・ロード・ブロック
４８３　制御ブロック
４８５　DSPプロセッサ初期設定ファイル

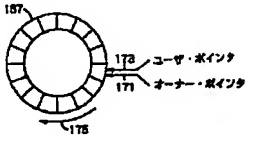
【図１２】



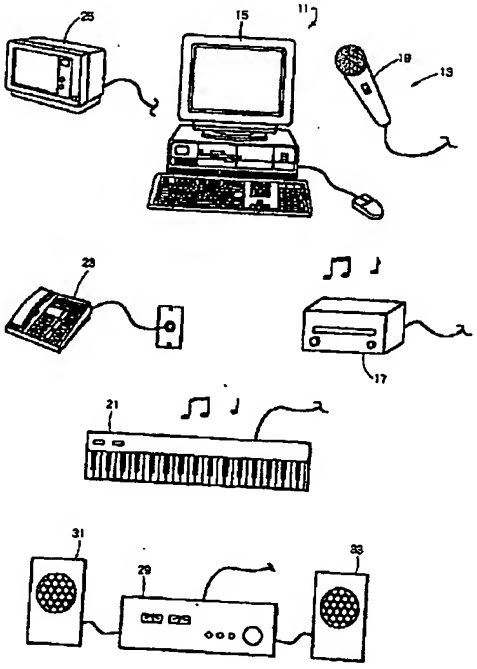
【図１３】



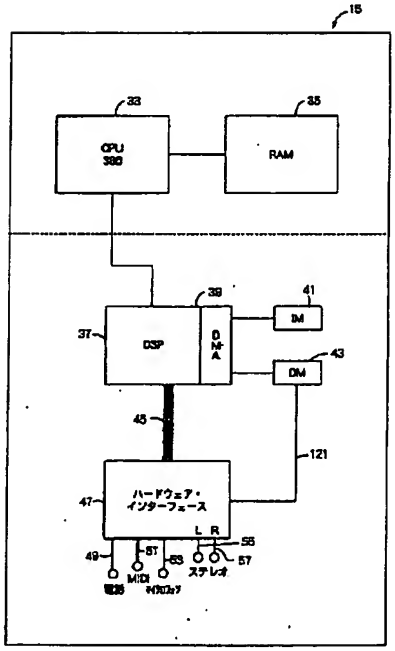
【図１６】



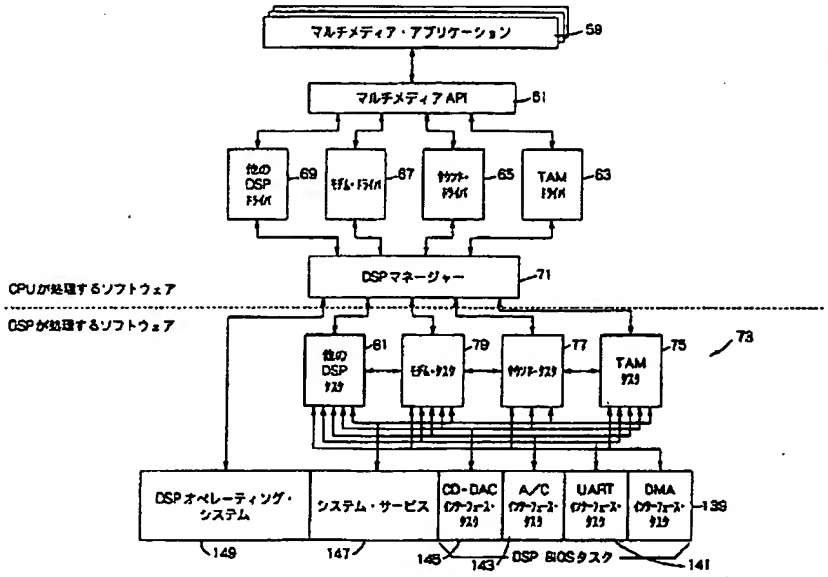
【図1】



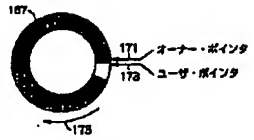
【図2】


15
33
CPU
35
RAM
39
41
IM
37
DSP
D
M
A
43
DM
45
121
47
ハードウェア・
インターフェース
L R
55
49
57
MIDI

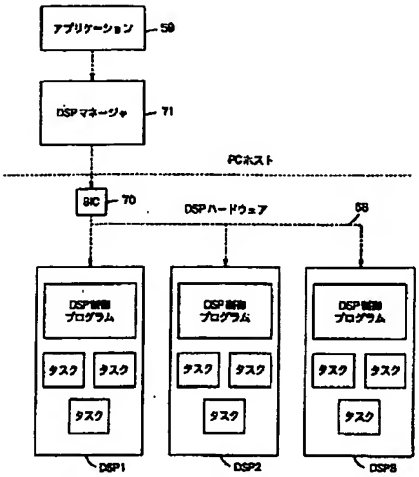
【図3】


マルチメディア・アプリケーション
59
マルチメディア API
61
他の DSP ドライバ
69
モデム・ドライバ
67
サンプル・ドライバ
65
TAM ドライバ
63
DSP マネージャー
71
CPUが処理するソフトウェア
DSPが処理するソフトウェア
他の DSP タスク
81
モデム・タスク
79
サンプル・タスク
77
TAM タスク
75
73
DSP オペレーティング・システム
149
システム・サービス
147
CD-DAC インターフェース・タスク
145
A/C インターフェース・タスク
143
UART インターフェース・タスク
DMA インターフェース・タスク
139
DSP BIOS タスク
141

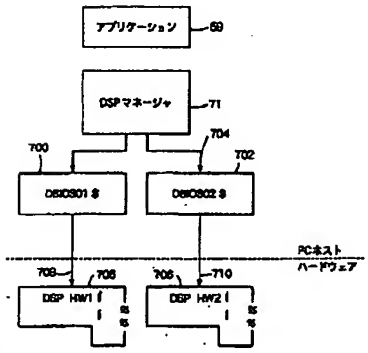
【図17】


167
171
オーナー・ポインタ
173
ユーザ・ポインタ
175

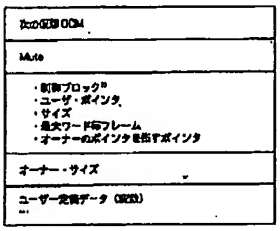
【図4】



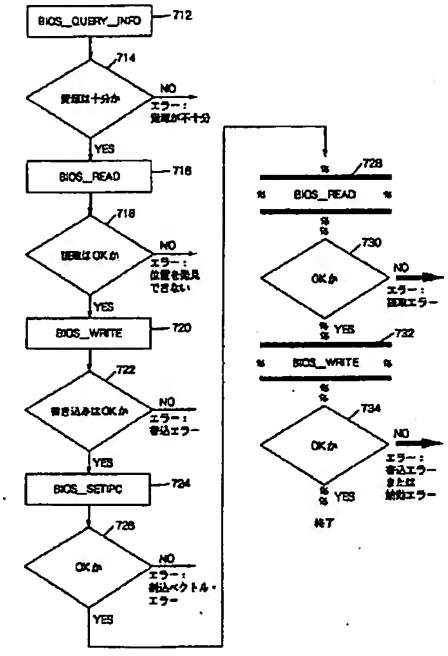
【図5】



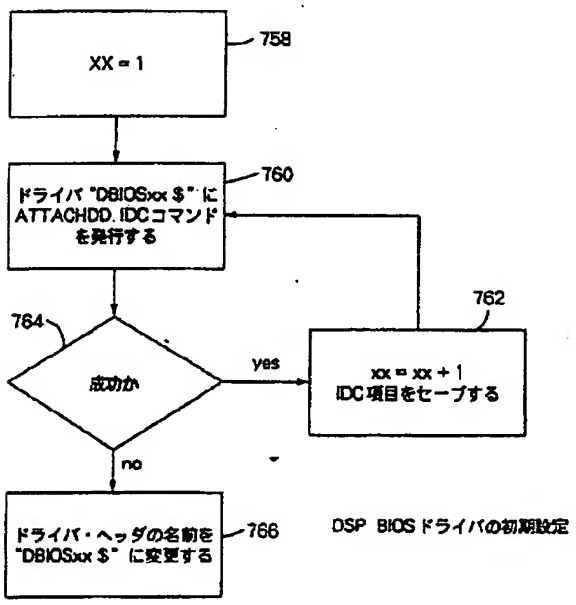
【図46】



【図6】

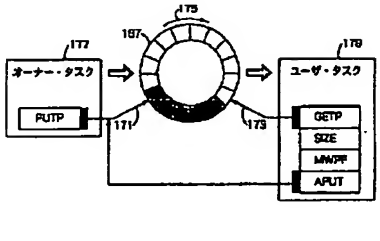


【図8】

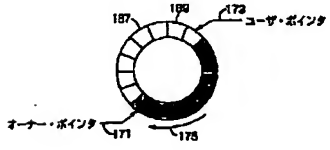


DSP BIOSドライバの初期設定

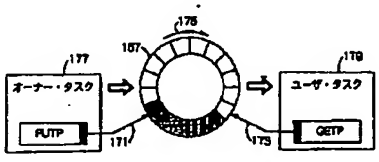
【図19】



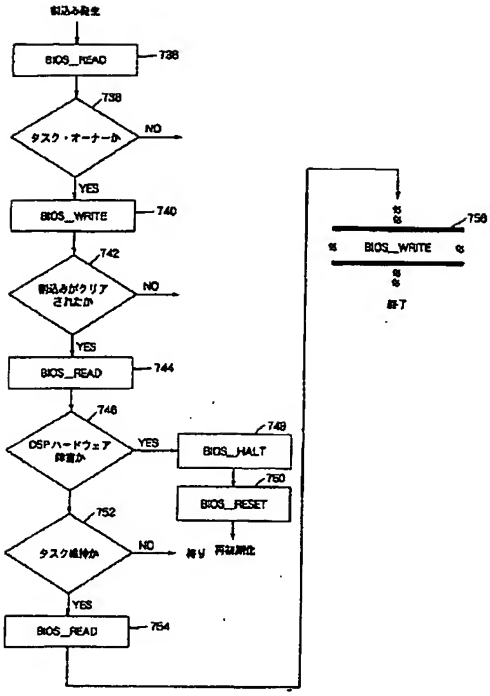
【図15】



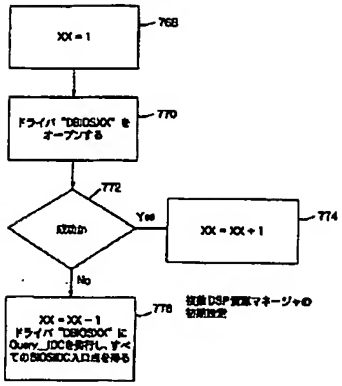
【図18】



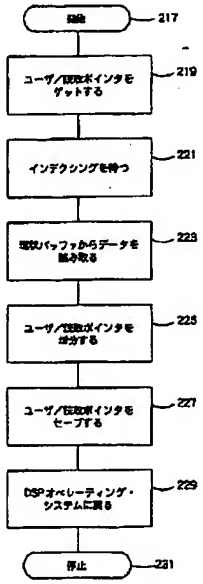
【図7】



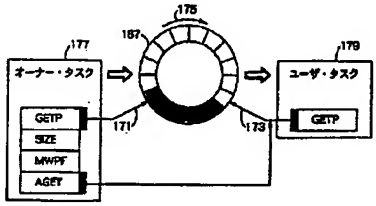
【図9】



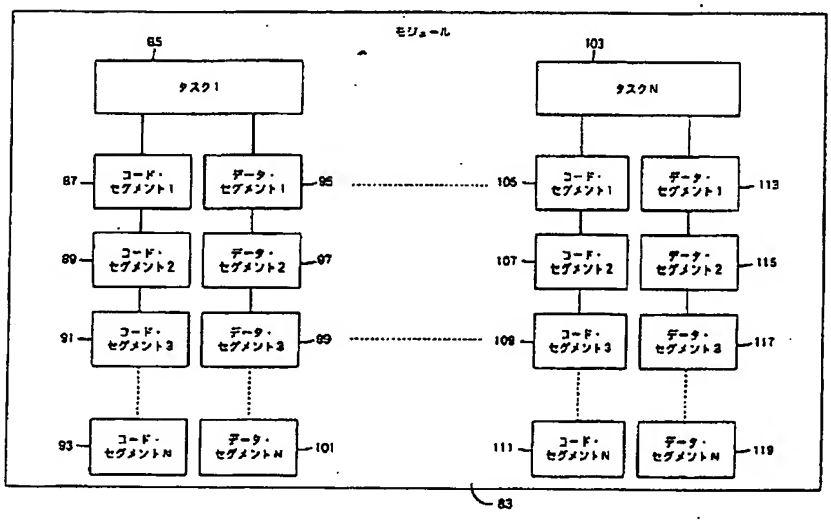
【図29】



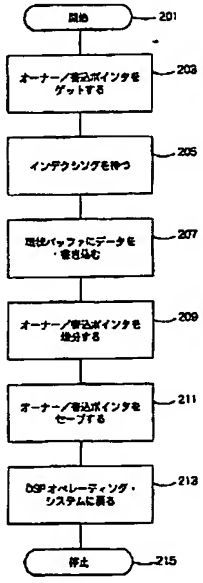
【図20】



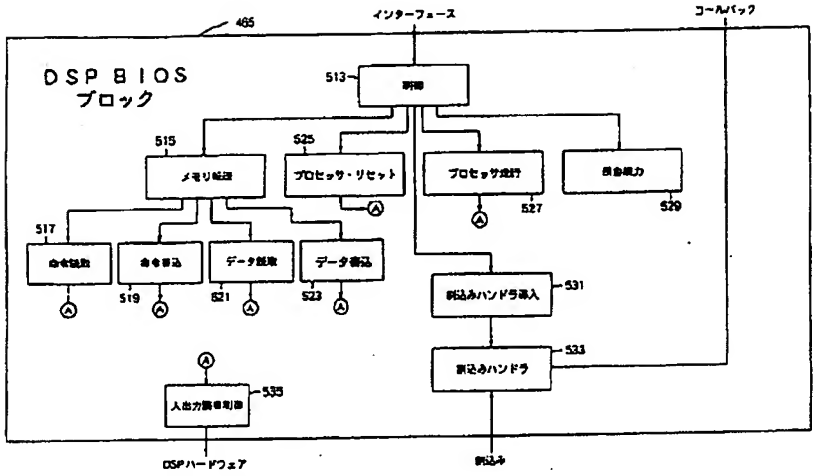
【図10】



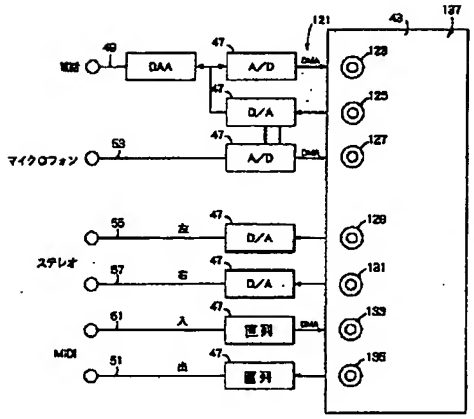
【図28】



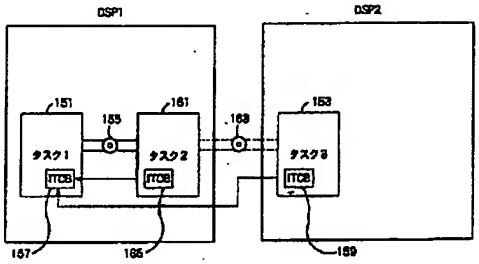
【図39】



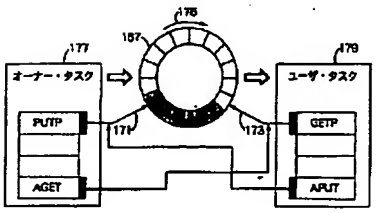
【図11】



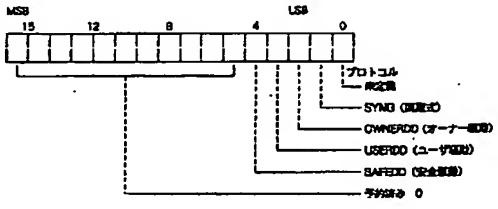
【図14】



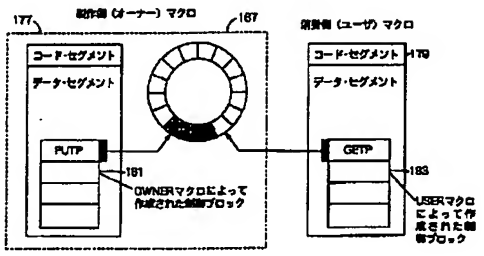
【図21】



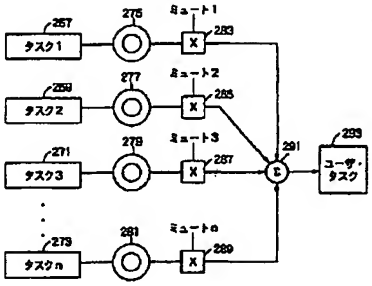
【図22】



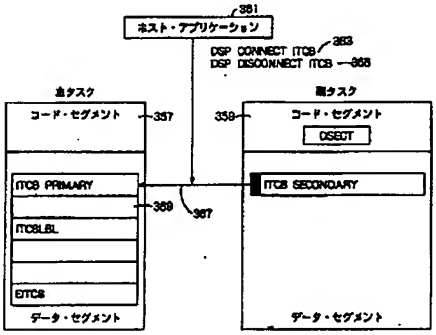
【図23】



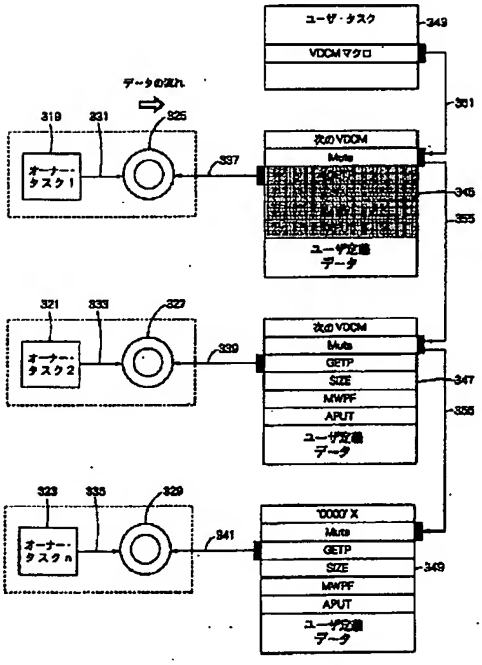
【図24】



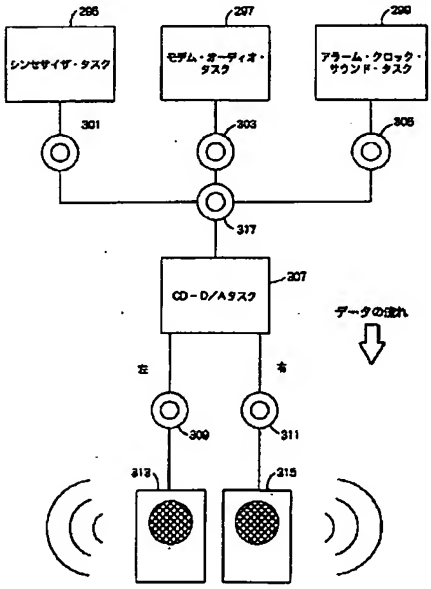
【図27】



【図25】



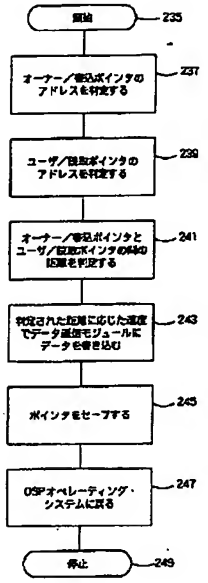
【図26】



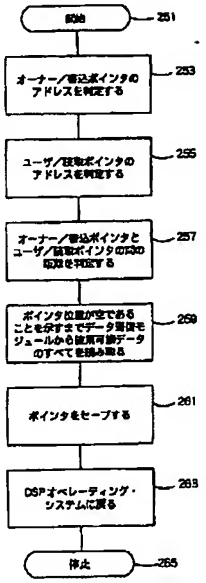
【図44】

```
label GPC (OWNER＝〈オーナー名〉| USER＝〈ユーザ名〉),
          {[size＝n | MINSIZE＝n],
          [PROTOCOL＝プロトコル],
          MAXWPF＝n,
          MPTRSEP＝n,
          [SAMPRATE＝n],
          [ADDRMODE＝アドレスモード],
          [MODE＝モード],
          {[MAXDELT＝n | MAXDELF＝n]},
          [STRIDE＝n | L]}
```

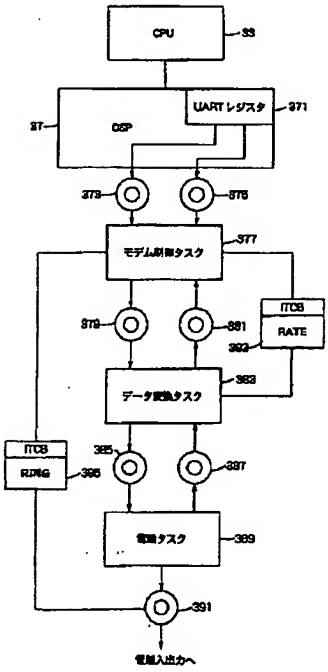
【図30】



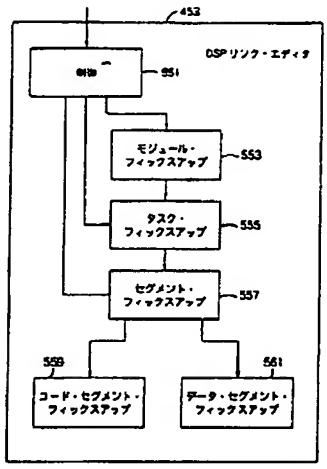
【図31】



【図32】



【図40】



【図47】

カテゴリー1
1) dsp Allocate See
2) dsp Initiation
3) dsp Load Module
4) dsp Load Task
5) dsp Load Thread

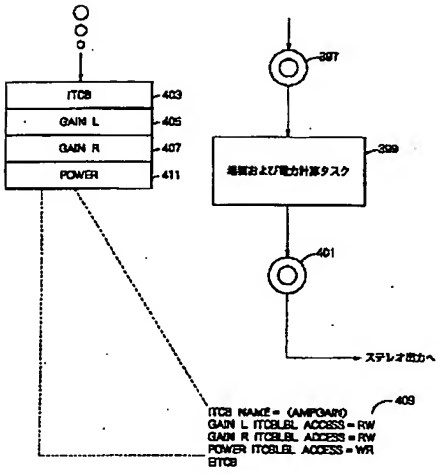
カテゴリー2
1) dsp Change CPF
2) dsp Change DMA
3) dsp Change Module State
4) dsp Change Task State
5) dsp connect DCM
6) dsp Connect ITCB
7) dsp Disconnect DCM
8) dsp disconnect ITCB
9) dsp Free Module
10) dsp Free Segment
11) dsp Free Task

カテゴリー3
1) dsp Memtransfer
2) dsp Reset
3) dsp Run

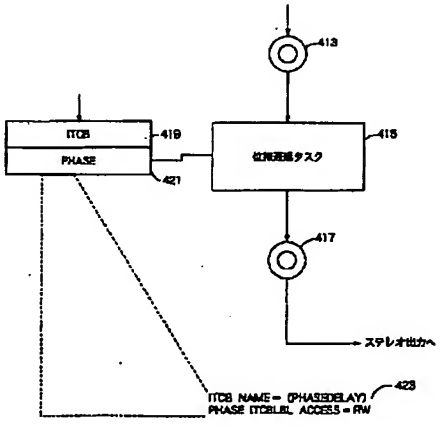
カテゴリー4
1) dsp Connect IPC
2) dsp Disconnect IPC

カテゴリー5
1) dsp Abilities
2) dsp Label to Address
3) dsp Name to Module Handle
4) dsp Name to Segment Handle
5) dsp Name to Thread Handle
6) dsp Query DSP Info
7) dsp Query Manager Info
8) dsp Query Misc Info
9) dsp Query Module Info

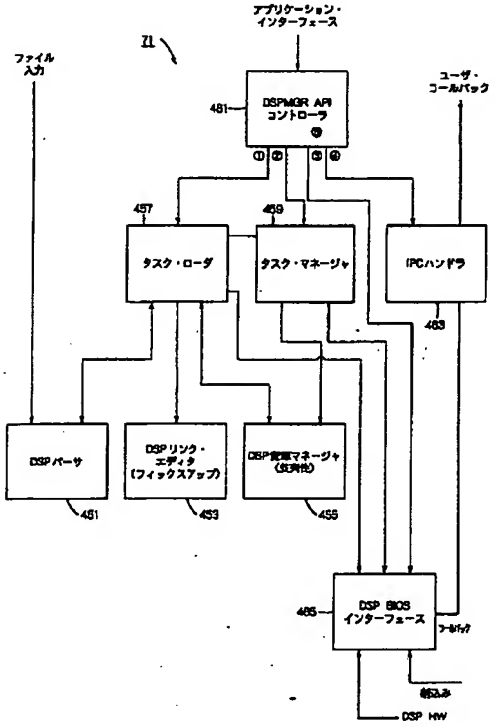
【図33】



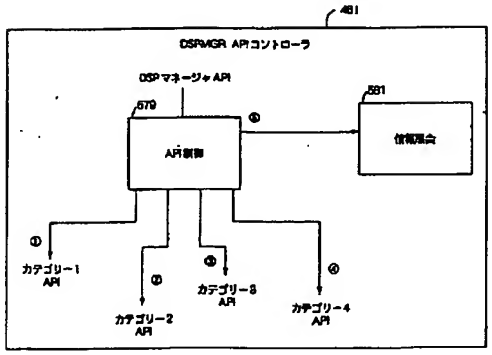
【図34】



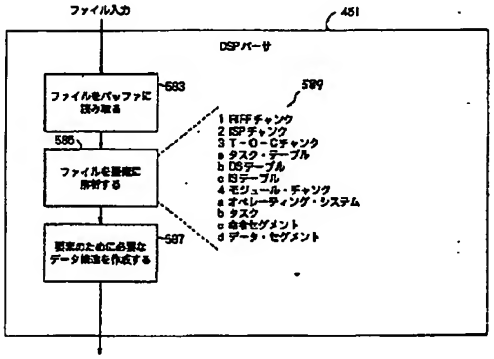
【図35】



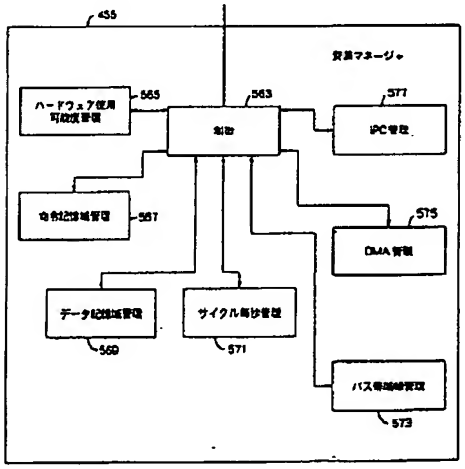
【図36】



【図38】



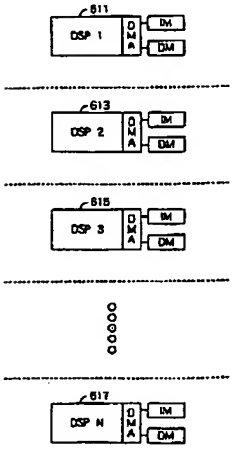
【図42】



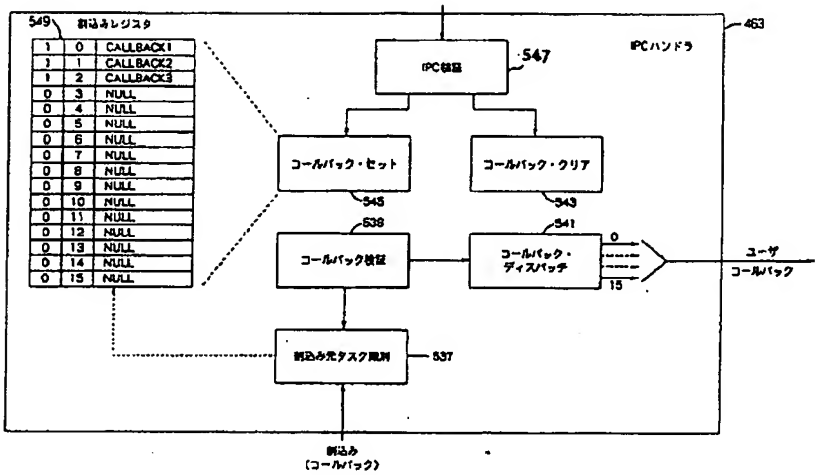
【図45】

オーナー・プロトコル	ユーザー・プロトコル			
	SYNC	OWNERDD	USERDD	SAFEDD
SYNC	•	•		
OWNERDD		•	•	
USERDD	•	•	•	•
SAFEDD		•		•

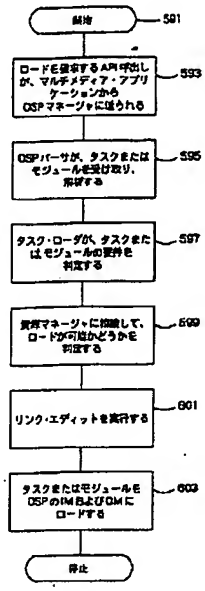
【図49】



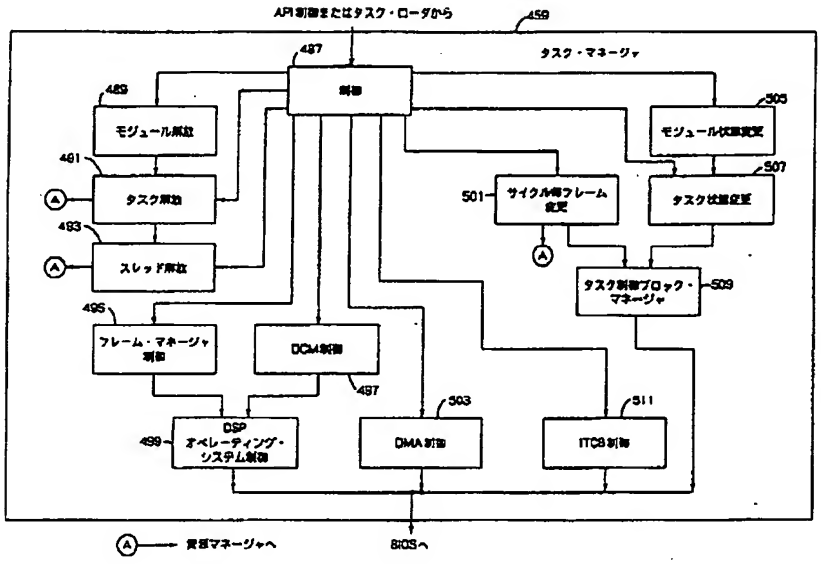
【図３７】



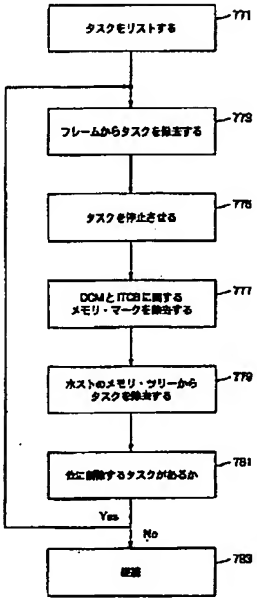
【図４８】



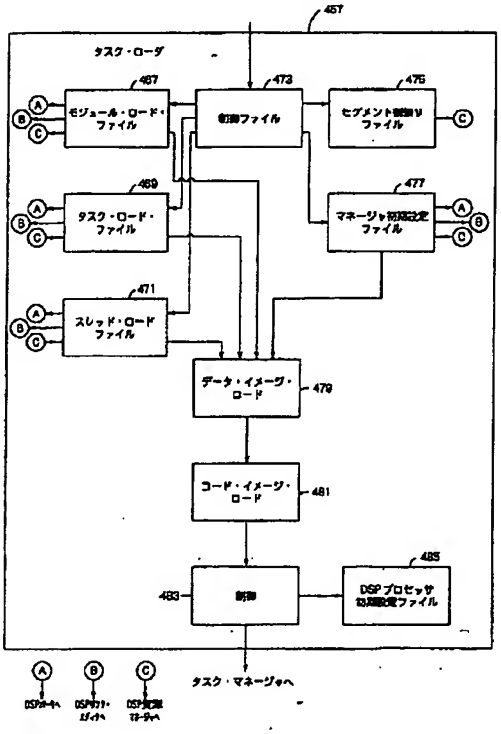
【図４１】



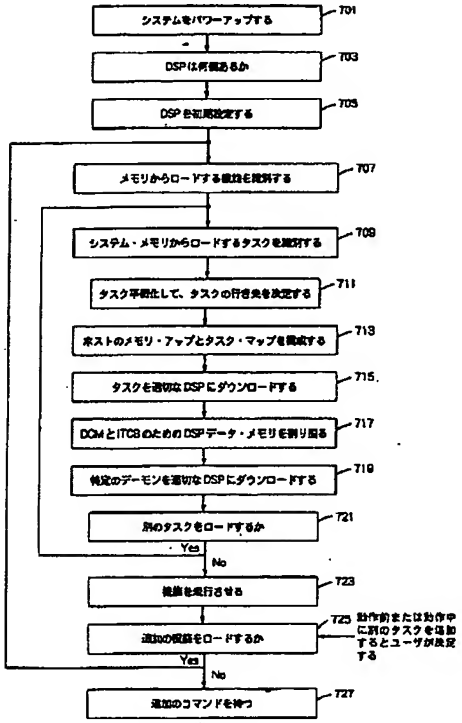
【図５２】



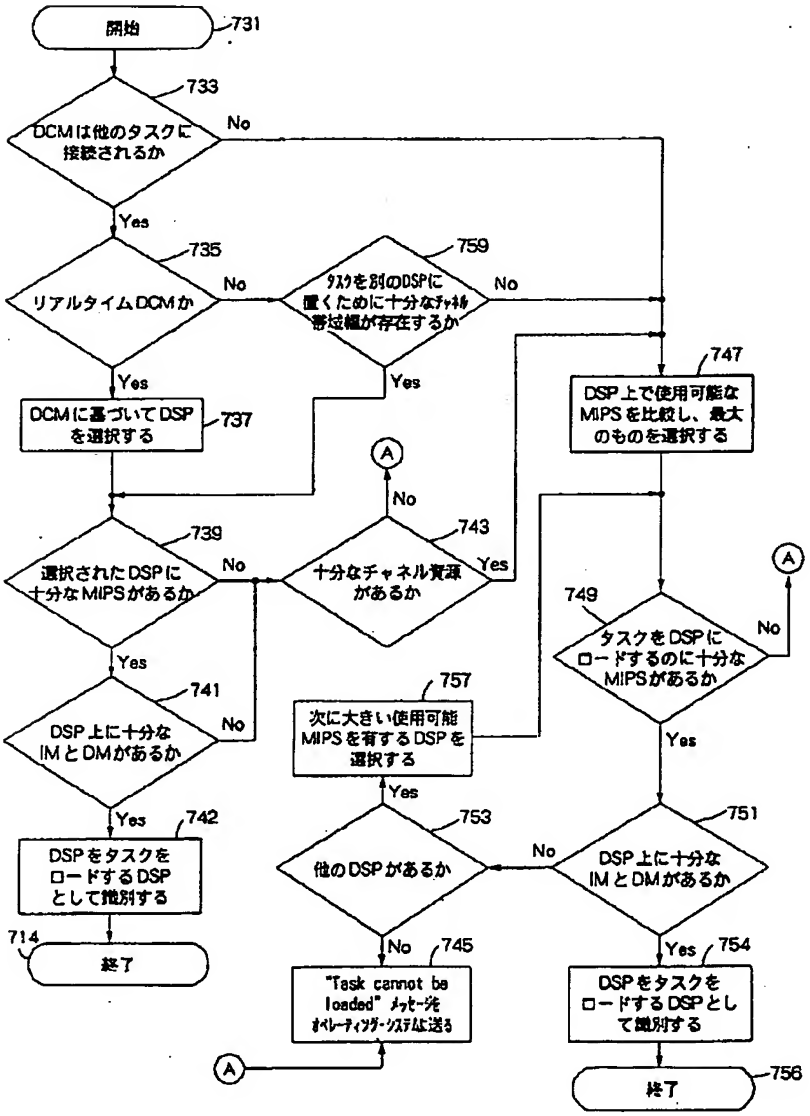
【図43】


457
タスク・ローダ
467
モジュール・ロード・ファイル
473
制御ファイル
475
セグメント制御ファイル
469
タスク・ロード・ファイル
477
マネージャ初期設定ファイル
471
スレッド・ロード・ファイル
479
データ・イメージ・ロード
481
コード・イメージ・ロード
483
制御
485
DSPプロセッサ初期設定ファイル
A
B
C
タスク・マネージャへ

【図50】


701
システムをパワーアップする
703
DSPは何個あるか
705
DSPを初期設定する
707
メモリからロードする個数を識別する
709
システム・メモリからロードするタスクを識別する
711
タスク平衡化して、タスクの行き先を決定する
713
ホストのメモリ・アップとタスク・マップを構成する
715
タスクを適切なDSPにダウンロードする
717
DCMとITCSのためのDSPデータ・メモリを割り当る
719
特定のデーモンを適切なDSPにダウンロードする
721
別のタスクをロードするか
Yes
No
723
実施を実行させる
725
追加の実施をロードするか
動作前または動作中に別のタスクを追加するとユーザが決定する
Yes
No
727
追加のコマンドを待つ

【図51】



フロントページの続き

(72)発明者　ホセ・エイ・エドゥアルテス
アメリカ合衆国33141　フロリダ州マイアミ・ビーチ　フェアウェイ・ドライブ　285

(72)発明者　ズイ・キュー・フィン
アメリカ合衆国33431　フロリダ州ボカ・ラトン　グリーンウッド・テラス　2600　ジー209

(72)発明者　ポール・アール・スィングル
アメリカ合衆国33445　フロリダ州デルレイ・ビーチ　ノース・ウエスト　ナインス・ストリート　3727

(72)発明者　サクソーン・ヨン
アメリカ合衆国33498　フロリダ州ボカ・ラトン　ワンハンドレッドアンドエイティーンシクスス・コートサウス　10299